Modellbasierte Spezifikation von RESTful SOA auf Basis flexibler SOM-Geschäftsprozessmodelle

Matthias Wolf
Universität Bamberg

Modellbasierte Spezifikation von RESTful SOA auf Basis flexibler SOM-Geschäftsprozessmodelle

Dissertation

zur Erlangung des akademischen Grades

doctor rerum politicarum (Dr. rer. pol.)

vorgelegt an der

Fakultät Wirtschaftsinformatik und

Angewandte Informatik der

Universität Bamberg

von

Herrn Dipl.-Wirtsch.Inf. Matthias Wolf

Bamberg, 10. April 2015

Gutachter:

Prof. Dr. Elmar J. Sinz

Prof. Dr. Sven Overhage

Tag der Disputation:

23. Juli 2015

Meiner Familie

Vorwort

Entwicklungsmethoden und dienstorientierte IT-Architekturen unterstützen die ganzheitliche Spezifikation von Anwendungssystemen auf Basis von Geschäftsprozessen. Handelt es sich jedoch um flexible Geschäftsprozesse, so führen deren Änderungen häufig zu einem verringerten Automatisierungsgrad und Inkonsistenzen im Informationssystem. Die Bereitstellung von Lösungen für die zielgerichtete Anpassung von Anwendungssystemen ist eine wichtige Herausforderung in der Systementwicklung. Während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Wirtschaftsinformatik, insbes. Systementwicklung und Datenbankanwendung, forschte ich an Methoden und IT-Architekturen zur modellbasierten Gestaltung und Pflege betrieblicher Informationssysteme. Das Ergebnis meiner Forschung ist die SOM-R-Methodik, einer modellgetriebenen Methodik für die ganzheitliche Entwicklung von RESTful SOA auf Basis flexibler SOM-Geschäftsprozessmodelle. Eine Vielzahl an Personen begleitete mich bei der Anfertigung dieser Arbeit mit ihrem Wissen, Ideen und Ratschlägen.

Mein ganz besonderer Dank gilt zuallererst meinem Doktorvater, Herrn Professor Dr. Elmar J. Sinz, der mir immer als unermüdlicher und geduldiger „Sparringspartner“ zur Seite stand. In zahllosen Gesprächen wurden die Ideen und Ergebnisse dieser Arbeit zielorientiert und konstruktiv diskutiert. Diese wertvolle Betreuung machte es erst möglich, die methodischen Ergebnisse meiner Forschung effizient zu erarbeiten und in den einheitlichen Rahmen einer Dissertation zu gießen.
Herrn Professor Dr. Overhage danke ich sehr herzlich für die Übernahme des Zweitgutachtens, sowie für seine wertvollen Hinweise während der Diskussionen zu meiner Arbeit. Herrn Professor Dr. Wirtz danke ich sehr für seine Mitgliedschaft in der Promotionskommission und seine Hinweise, die mir halfen verschiedene Realisierungsaspekte noch schärfer zu fokussieren.

Mein besonderer Dank gilt einer Reihe von Kollegen. Durch seine freundschaftliche Unterstützung und konstruktiven Ideen trug Herr Thomas Benker wesentlich zur Qualitätssicherung von wichtigen Ergebnissen der vorliegenden Arbeit bei. Frau Dr. Beate Hartmann sprach mir viele Male Motivation zu und gab mir zahllose Hinweise für die Anfertigung der Dissertation. Die Ratschläge von Herrn Armin Duske halfen mir immer wieder bei der Konzentration auf den Kern der Arbeit. Herrn Felix Härer, Herrn Carsten Jürck und Herrn Andree Teusch danke ich sehr für ihre Hinweise und Hilfestellungen.
Im freundschaftlichen Umfeld des Lehrstuhls fiel es mir sehr leicht, mit Spaß und fokussiert zu arbeiten. Allen Kollegen, die mich während dieser Zeit begleitet haben, möchte ich hierfür sehr danken. Meinen Kollegen von CEUS danke ich darüber hinaus für ihre herzliche Unterstützung.

Meinen Eltern Irmlind und Karl-Joachim, sowie meinem Bruder Sebastian und meiner Frau Katrin danke ich für die sehr hilfreichen Ratschläge, die Ermutigungen sowie die Durchsicht des Manuskripts. Meine Frau Katrin gab mir durch ihre Liebe und die nötigen Ablenkungen immer wieder neue Kraft. Erst der Rückhalt und die Unterstützung meiner ganzen Familie ermöglichten den erfolgreichen Abschluss dieser Dissertation.

Bamberg, September 2015

Matthias Wolf

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

ADK	Anwendungsteil - Datenverwaltungsteil - Kommunikationsteil
API	Application Programming Interface
AT	Aufgabenträger
ATL	Atlas Transformation Language
AwS	Anwendungssystem
bA	Betriebliche Aufgabe
BMM	Beziehungsmetamodell
bO	Betriebliches Objekt
BPEL	Business Process Execution Language (siehe auch WS-BPEL)
BPMN	Business Process Model and Notation
BS	Basissystem
bT	Betriebliche Transaktion
CCK	Computer-Computer-Kommunikation
CIM	Computational Independent Model
CRUD	Create - Read - Update - Delete
DB	Datenbank
DBS	Datenbanksystem
DBVS	Datenbankverwaltungssystem
DMS	Datenmanagementsystem
DOT	Datenobjekttyp
DSL	Domain Specific Language
DT	Dokumenttyp
ED	Entitätsdienst
EMF	Eclipse Modeling Framework
EMP	Eclipse Modeling Project
EOT	Entitätsspezifischer Objekttyp
EPK	Ereignisgesteuerte Prozesskette
ER	Entitätsressource
ER-Typ	Entity-Relationship-Typ
ESB	Enterprise Service Bus
E-Typ	Entity-Typ
eVOT	Elementarer Vorgangsobjekttyp
eVR	Elementare Vorgangsressource
GP	Geschäftsprozess
GPM	Geschäftsprozessmodell
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
IAS	Interaktionsschema

IOS	Interface-Objektschema
IOT	Interface-Objekttyp
IP	Internet Protocol
IS	Betriebliches Informationssystem
IT	Informationstechnologie
Java SE	Java Standard Edition
JAXB	Java Architecture for XML Binding
JAX-RS	Java API for RESTful Web Services
JET	Java Emitter Templates
JSON	JavaScript Object Notation
JSR	Java Specification Request
KOS	Konzeptuelles Objektschema
KOS-P	Konzeptuelles Objektschema persistenter Klassen
KOT	Konzeptueller Objekttyp
LOT	Leistungsspezifischer Objekttyp
MCK	Mensch-Computer-Kommunikation
MDA	Model Driven Architecture
MDD	Model Driven Development
MDE	Model Driven Engineering
MDSD	Model Driven Software Development
MEP	Message Exchange Pattern
MM	Metamodell
MVC	Model-View-Controller
M2C	Model-to-Code-Transformation
M2M	Model-to-Model-Transformation
neVOT	Nicht-elementarer Vorgangsobjekttyp
neVR	Nicht-elementare Vorgangsressource
NoSQL	No SQL / Not only SQL
OMG	Object Management Group
OOT	Objektspezifischer Objekttyp
O/R-Mapping	Objekt/Relationales Mapping
OS-ED	Objektschema der Entitätsdienste
OS-VD	Objektschema der Vorgangsdienste
PIM	Platform Independent Model
PM	Platform Model
PSM	Platform Specific Model
QVT	Query-View-Transformation
rDBVS	Relationales Datenbankverwaltungssystem
REST	REpresentational State Transfer
R-IOT	Interface-Objekttyp (Modellebene REST-Architektur)
ROA	Resource-oriented Architecture
RS-ER	Ressourcenschema der Entitätsressourcen
RS-VR	Ressourcenschema der Vorgangsressourcen
R-Typ	Relationship-Typ

SERM	Strukturiertes Entity-Relationship-Modell
SOA	Service-oriented Architecture
SOAP	Eigenname, ursprünglich: Simple Object Access Protocol
SOM	Semantisches Objektmodell
SQL	Structured Query Language
TOT	Transaktionsspezifischer Objekttyp
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VD	Vorgangsdienst
VES	Vorgangs-Ereignis-Schema
V-Modell	Vorgehensmodell des Semantischen Objektmodells
VOS	Vorgangsobjektschema
VOT	Vorgangsobjekttyp
VR	Vorgangsressource
WADL	Web Application Description Language
WfMS	Workflow-Management-System
WS*	Web-Service-Technologien
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Description Language
WWW	World Wide Web
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 Einleitung

1.1 Motivation und Problemstellung

Unternehmen agieren in einer dynamischen Welt von stetig wachsender Komplexität. In einem solchen Umfeld ist die kontinuierliche Anpassung der betrieblichen Leistungserstellung an sich verändernde Wettbewerbsbedingungen und Umwelteinflüsse eine notwendige Konsequenz, um das Überleben von Unternehmen sicherzustellen (z. B. [WSL+11b, S. 808 ff.], [DBW+10]). Das Ziel der Anpassung ist einerseits die Weiterentwicklung von bestehenden, und andererseits die Gestaltung neuer Geschäftsprozesse, um die betriebliche Leistungserstellung auf die wettbewerbskritischen Anforderungen auszurichten. Ein zentraler Erfolgsfaktor bei der evolutionären Anpassung betrieblicher Systeme ist die Flexibilität von Geschäftsprozessen (z. B. [Moos09, S. 56 f.], [KaBl05b, S. 4], [Meff85, S. 121 ff.]). Ein flexibler Geschäftsprozess verändert seine Struktur und/oder sein Verhalten, um damit der Herausforderung des stetigen Wandels zu begegnen ([WSL+11a, S. 88], [BBF+11, S. 2]).

Die Änderungen von flexiblen Geschäftsprozessen führen meist zu einer kontinuierlichen Verringerung des Automatisierungsgrades im informationsverarbeitenden Teil eines betrieblichen Systems. Es entstehen Entwicklungsrückstände der eingesetzten Anwendungssysteme (AwS) in Bezug auf die, in den Geschäftsprozessen spezifizierten, fachlichen Anforderungen (z. B. [SBB+11, S. V], [NSA14, S. 41 f.]). Fachlich begründete Änderungen führen somit zu Inkonsistenzen zwischen der Aufgaben- und der Aufgabenträgerebene des betrieblichen Informationssystems (IS) (z. B. [WSL+11a, S. 79 f.], [HRO+10, S. 3], [BER+10], [TuLa05]). Die (Wieder-)Herstellung eines konsistent beschriebenen IS erfordert die Anwendung eines ganzheitlichen Lösungsansatzes, um AwS immer wieder zeitnah auf die flexiblen Geschäftsprozesse ausrichten zu können.

Die Gestaltung und Pflege von betrieblichen Informationssystemen ist Aufgabe der Systementwicklung. Diese sieht sich im Kontext der vorliegenden Arbeit der Herausforderung gegenüber, Methoden und Architekturen bereitzustellen, die die Entwicklung flexibler AwS sowie deren rasche und konsistente Anpassung an die Änderungen flexibler Geschäftsprozesse unterstützen. Hieraus ergibt sich die Problemstellung der Arbeit, die einleitend in den folgenden zwei Fragestellungen formuliert wird:

- Welche methodischen Hilfsmittel unterstützen die Bewältigung der Komplexität einer systematischen Überführung veränderter Anforderungen eines flexiblen Geschäftsprozesses in eine (darauf) angepasste AwS-Spezifikation, und damit die evolutionäre Weiterentwicklung eines betrieblichen Informationssystems auf seinen unterschiedlichen Beschreibungsebenen? (erste Frage)
- Welche Systemarchitektur unterstützt die schrittweise Anpassung von AwS und deren konsistente Abstimmung mit den flexiblen Geschäftsprozessen? (zweite Frage)

Die Entwicklung von AwS, die eine rasche Anpassung an sich ändernde Anforderungen flexibler Geschäftsprozesse unterstützen, erfordert die Bereitstellung von Methoden, um betriebliche

Informationssysteme ganzheitlich spezifizieren zu können (erste Frage). Die Untersuchung von Konzepten, Modellen und Methoden für die Analyse, Gestaltung und Pflege von betrieblichen Informationssystemen ist zentraler Forschungsgegenstand der Wirtschaftsinformatik (z. B. [ÖWB10, S. 3 f.]). Modelle sind hierbei das wichtigste methodische Hilfsmittel zur Reduzierung der, im Allgemeinen, hohen Komplexität im Betrachtungsgegenstand „Informationssystem“ ([FSF+14, S. 49], [FeSi13, S. 133]). In jüngster Zeit fokussieren Lösungsansätze aus diesen Themenbereichen die Konstruktion modellgetriebener Methoden, deren verfolgte Zielsetzung eine (teil-)automatisierte Durchführung der Systementwicklungsaufgabe ist (z. B. [FrRu07], [SVE+07], [CMW09]). Der Einsatz von Modellen dient dabei nicht nur dem Zweck der Analyse und Dokumentation von IS. Vielmehr entsteht durch die Gestaltung von Modellen das zentrale Artefakt für die automatisierte Ausführung eines Systemerzeugungsprozesses.

Neben dem Einsatz methodischer Hilfsmittel leisten flexible IT-Systeme einen entscheidenden Beitrag, um eine Anpassung von AwS an die veränderten Anforderungen in flexiblen Geschäftsprozessen zu fördern (zweite Frage). Seit vielen Jahren wird im Bereich des Software Engineerings versucht, weitere Flexibilitätspotenziale durch die Nutzung der Technologien und Architekturen service-orientierter IT-Systeme zu erschließen (z. B. [BBF+11, S. 8 f.], [HRO+10], [Melz10, S. 33 ff.]). Kernkonzept von service-orientierten Architekturen (SOA) ist die Bereitstellung lose gekoppelter Dienste, welche die Durchführung von Geschäftsprozessaufgaben unterstützen (z. B. [BWB11, S. 187], [Star11, S. 314 ff.], [Grah08, S. 8 f.]). An den Entwurf und die Realisierung von SOA sind im Allgemeinen hohe Erwartungen bezüglich der Anpassbarkeit und Interoperabilität ihrer verteilten Komponenten geknüpft (z. B. [FeSi13, S. 495], [BWB11, S. 187], [ERM10, S. 18], [Melz10, S. 38 ff.], [Erl08b, S. 55 ff.], [StTi07b, S. 18]). In jüngster Zeit rückt der Architekturstil *REpresentational State Transfer* (REST, [Fiel00]) als Realisierungsform von SOA (RESTful SOA) zunehmend in den Fokus der Betrachtung (z. B. [ECP+13], [Tilk11], [RiRu07]). Die Vorteile des Einsatzes von REST werden im betrieblichen Umfeld dabei hinsichtlich Flexibilität, Interoperabilität und Skalierbarkeit für ad-hoc Integrationsszenarien webbasierter Anwendungssysteme gesehen (z. B. [PZL08, S. 13], [Tilk11, S. 207 ff.], [RiRu07, S. 299 ff.]). REST und SOA bilden in dieser Arbeit die Zielarchitektur des Systementwicklungsprozesses (vgl. Abschnitt 1.2.1).

In den Themenbereichen der modellgetriebenen Entwicklung, Spezifikation von RESTful SOA und Weiterentwicklung von Systemen bestehen im aktuellen Forschungsstand eine Reihe von Defiziten in Bezug auf die tiefergehende Betrachtung und die Lösung der vorgestellten Problemstellung (vgl. Kapitel 8).

- Im Bereich der modellgetriebenen Entwicklung behandelt eine Vielzahl an Ansätzen die Überführung von Workflow- und auch Geschäftsprozessmodellen in SOA-Spezifikationen (z. B. [MRS11], [DRG+10b], [HPF10], [TLD+07], [GTP07], [Rick07]). Dabei ist in diesen Arbeiten generell eine Fokussierung auf niedrigere, meist softwaretechnische Abstraktionsebenen feststellbar. Eine ganzheitliche Betrachtung der Entwicklung von Informationssystemen erfordert jedoch die Einbeziehung der plattformunabhängigen Aufgabenebene der Geschäftsprozesse. In aktuellen modellgetriebenen Arbeiten wird die Verringerung der semantischen Lücke zwischen Aufgabenebene und plattformspezifischer Anwendungsebene methodisch häufig nicht adäquat behandelt ([FrRu07,

S. 40 f.], [TeSi12, S. 1646]). Dies gilt insbesondere für die durchgängig modellbasierte Überführung von Geschäftsprozessmodellen in konkrete Realisierungsformen von SOA, z. B. in RESTful SOA.

- Die Fokussierung auf die verhaltensorientierte Sicht von Geschäftsprozessmodellen ist in vielen modellbasierten Systementwicklungsansätzen als weiteres Defizit erkennbar (z. B. [DRG+10b], [Rick07], [TLD+07]). Dabei wird vor allem die Überführung von Spezifikationen der Struktursicht von Geschäftsprozessmodellen in die Komponenten von SOA nur unzureichend thematisiert. Für eine ganzheitliche Entwicklung betrieblicher Informationssysteme, ihre Anpassung an veränderte Anforderungen und die Konzeption von Lösungsansätzen zur Bewältigung der damit einhergehenden Komplexität ist jedoch die Einnahme einer strukturorientierten Perspektive auf die Beschreibungsebenen erforderlich. Auch hier besteht weiterer Forschungsbedarf.
- Eine grundsätzliche Herausforderung der Systementwicklung im Allgemeinen, als auch der modellgetriebenen Systementwicklung im Speziellen, ist die Weiterentwicklung betrieblicher Anwendungssysteme, um diese wieder konsistent auf geänderte fachliche Anforderungen auszurichten (z. B. [FrRu07, S. 41], [Somm12, S. 276 ff.], [BeRa00, S. 76 ff.]). Generell lässt sich in diesem Bereich eine Konzentration vieler modellgetriebener Ansätze auf die „Neu“-Entwicklung von Anwendungssystemen feststellen (z. B. [RRS+06], [TLD+07], [GTP07]). Die AwS-Weiterentwicklung wird hier meist nicht, oder nur defizitär, behandelt, z. B. im Rahmen einer automatisierten Systemerzeugung auf plattformnahen Modellebenen (z. B. [FrRu07, S. 41], [SVE+07, S. 11 ff.], [PeMe06, S. 38]). Die Überwindung der semantischen Lücke zwischen der Aufgabenebene von flexiblen Geschäftsprozessmodellen und der plattformspezifischen Anwendungsebene von SOA ist jedoch nicht vollständig automatisiert beschreibbar und setzt das Treffen von Entwurfsentscheidungen durch den „kreativen“ Systementwickler voraus. Die Einbeziehung dieser personellen, und damit nicht-automatisierbaren Entwicklungsschritte stellt eine Herausforderung dar, zu deren Behandlung die modellgetriebenen Methoden für die aufgespannte Problemstellung bislang keine methodisch durchgängigen Lösungsansätze liefern.

Die Durchführung der vorliegenden Systementwicklungsaufgabe dieser Arbeit setzt die Verfügbarkeit methodischer Hilfsmittel voraus, welche die Bewältigung der bestehenden Komplexität im betrieblichen Informationssystem, die zielgerichtete Anpassung des flexiblen (serviceorientierten) AwS an veränderte Anforderungen flexibler Geschäftsprozessmodelle und die Überbrückung der zwischen diesen Abstraktionsebenen bestehenden semantischen Lücke unterstützen. Die Bereitstellung von Methoden für die ganzheitliche Gestaltung und Pflege von IS ist eine bisher nur unzureichend gelöste Herausforderung und ein zentraler Forschungsgegenstand der Systementwicklung. Der Beitrag der vorliegenden Arbeit besteht in der Konstruktion einer Entwicklungsmethodik zur Verringerung der vorgestellten methodischen Defizite.

1.2 Untersuchungsproblem der Arbeit

Die Abgrenzung der Problemstellung und die Beschreibung der Zielsetzung dieser Arbeit erfolgen anhand des Begriffs des *Untersuchungsproblems* [Fers79, S. 43 ff.]. Das Unter-

suchungsproblem besteht aus einem *Untersuchungsobjekt*, das die Eigenschaften des betrachteten Forschungsgegenstandes bestimmt, sowie dem mit dem Forschungsvorhaben verfolgten *Untersuchungsziel*. Nachfolgend werden Untersuchungsobjekt (Abschnitt 1.2.1) und Untersuchungsziel (Abschnitt 1.2.2) der Arbeit eingeführt und erläutert.

1.2.1 Untersuchungsobjekt

Die Abgrenzung des Untersuchungsobjekts erfolgt durch die Bestimmung seiner Eigenschaften, die zugleich die Restriktionen für die Konstruktion des Lösungsansatzes der Arbeit festlegen.

Modellbasierte Spezifikation auf Basis der SOM-Methodik
Die umfassende Behandlung des vorliegenden Untersuchungsproblems erfordert einen Ansatz zur ganzheitlichen Modellierung betrieblicher Informationssysteme. In dieser Arbeit beschreibt die *SOM-Methodik* [FeSi13, S. 194 ff.] die Basis für die Erarbeitung einer Lösung für das vorgestellte Konstruktionsproblem. Die SOM-Methodik definiert einen Architekturrahmen sowie ein einheitliches Begriffssystem, welche das methodische Fundament für die zu konstruierende Entwicklungsmethodik bilden. Die Modellierung einer struktur- und verhaltensorientierten Sicht auf Geschäftsprozesse erfolgt mit dem objekt- und geschäftsprozessorientierten SOM-Ansatz. Die durchgängige objektorientierte Ausrichtung der SOM-Methodik ist ein weiteres Merkmal, das ihre Eignung für die modellbasierte Spezifikation von RESTful SOA, ausgehend von Geschäftsprozessmodellen, bestätigt. Den Einsatz der SOM-Methodik im Rahmen der modellbasierten Spezifikation von Anwendungssystemen zeigt eine Reihe von Beiträgen (z. B. [PüSi10], [TeSi12], [BeJa09]).

Abgrenzung: Eine Diskussion alternativer Ansätze zur Modellierung von Geschäftsprozessen, wie ereignisgesteuerte Prozessketten (EPK, [KNS92]) oder die Business Process Model and Notation (BPMN, [OMG11a]), ist nicht Gegenstand der Arbeit.

Strukturflexible Geschäftsprozesse als Ausgangspunkt der Systementwicklung
Im Fokus dieser Arbeit stehen die Entwicklung und Weiterentwicklung von AwS zur Unterstützung von Geschäftsprozessen, deren Struktur an interne oder externe Systemveränderungen angepasst wird. Der Auslöser des Systementwicklungsprozesses ist somit ein strukturflexibler Geschäftsprozess, der als SOM-Geschäftsprozessmodell modelliert wird. Des Weiteren wird das Untersuchungsobjekt dahingehend restringiert, dass die Änderungen am Geschäftsprozessmodell schrittweise erfolgen.

Abgrenzung: Geschäftsprozesse können neben Strukturflexibilität auch eine Verhaltensflexibilität aufweisen [BBF+11, S. 31 f.]. Die vorliegende Arbeit untersucht diesbezüglich nur die Anpassung von existierenden Anwendungssystemen, um diese mit den veränderten Anforderungen strukturflexibler SOM-Geschäftsprozessmodelle abzustimmen. Nicht behandelt werden die Gestaltung verhaltensflexibler Geschäftsprozessmodelle und ihre unterstützenden AwS. Die Änderungen an einem Geschäftsprozessmodell erfolgen schrittweise durch die Anpassung einzelner Modellbausteine. Die Bewältigung der Manipulation eines ganzen Teilprozesses, der eine Vielzahl an Bausteinen umfasst, wird dagegen nicht explizit untersucht. Des Weiteren liegt die Aufgabe der eigentlichen Erstellung der SOM-Geschäftsprozessmodelle außerhalb des Betrachtungsfokus dieser Arbeit. Die Änderungen an der Modellstruktur repräsentieren das

Ergebnis des Anpassungsprozesses, mit dem das betriebliche System die Dynamik der betrieblichen Umwelt vorwegnimmt oder auf diese reagiert [WSL+11a, S. 88].

RESTful SOA als Architektur flexibler Anwendungssysteme
Die Unterstützung der flexiblen Geschäftsprozesse soll vollständig durch ein neu-entwickeltes Anwendungssystem erfolgen. Um das Untersuchungsobjekt diesbezüglich durchgängig behandeln zu können, ist die Wahl einer konkreten Zielarchitektur auf der softwaretechnischen Ebene des IS erforderlich. Die Realisierungsform der flexiblen AwS ist in dieser Arbeit die RESTful SOA, die damit die zu spezifizierende Zielarchitektur des Entwicklungsprozesses dargestellt. RESTful SOA erscheinen aufgrund ihrer Merkmale der Ressourcen- und Dokumentenorientierung geeignet, die Entwicklung von Struktur- und Verhaltensmerkmalen ihrer Komponenten aus den Bestandteilen von SOM-Geschäftsprozessmodellen im Rahmen der konzeptuellen Modellierung behandeln zu können (Abschnitt 3.3). Durch die Bestimmung von Abhängigkeiten zwischen den modellierten Bausteinen auf den Ebenen des IS erscheint eine Anhebung der Abstraktionsebene der gesamten Systementwicklungsaufgabe möglich.

Abgrenzung: Der Auslöser für die Entwicklung bzw. Anpassung des Anwendungssystems ist stets fachlicher Druck aufgrund von Änderungen auf der Geschäftsprozessebene. Demzufolge liegen das Reengineering bestehender AwS sowie die Weiterentwicklung von AwS aufgrund geänderter technischer Rahmenbedingungen außerhalb des Fokus der vorliegenden Arbeit. Die Spezifikation der Systemimplementierung stellt hierbei das Ergebnis einer durchgeführten Entwicklung in dieser Arbeit dar. Die Aufgaben der Einführung oder Versionierung des entwickelten Systems werden nicht behandelt.

Neben RESTful SOA werden in der Literatur alternative Technologien zur Unterstützung flexibler Geschäftsprozesse diskutiert. Beispiele hierfür sind SOA auf Basis von Web-Service-Technologien (WS*-SOA), Workflowmanagementsysteme oder auch ereignisgetriebene Architekturen (z. B. [PZL08], [Romm08], [StTi07a]). Eine tiefergehende Analyse und ein Vergleich dieser Alternativen mit RESTful SOA sind nicht Gegenstand dieser Arbeit.

Das **Untersuchungsobjekt** der Forschungsarbeit lässt sich wie folgt zusammenfassen. Es ist definiert als die *modellbasierte Spezifikation betrieblicher Informationssysteme, deren Anwendungssysteme als RESTful SOA realisiert werden. Die Aufgaben des Informationssystems werden in Form von SOM-Geschäftsprozessmodellen beschrieben.*

1.2.2 Untersuchungsziel

Das Untersuchungsziel dieser Forschungsarbeit ist die *Konstruktion einer Entwicklungsmethodik zur modellbasierten Spezifikation von RESTful SOA auf Basis von strukturflexiblen SOM-Geschäftsprozessmodellen*.

Zur Bewältigung ihrer Komplexität wird diese Zielsetzung in zwei Teilziele untergliedert. Das *Untersuchungsziel* ist demnach die Konstruktion einer Entwicklungsmethodik für die

- modellbasierte Gestaltung von RESTful SOA auf Basis von SOM-Geschäftsprozessmodellen (erstes Teilziel), sowie die

- modellbasierte Pflege von RESTful SOA durch die Anpassung ihrer Komponenten an veränderte fachliche Anforderungen strukturflexibler SOM-Geschäftsprozessmodelle (zweites Teilziel).

Die Methodik unterstützt die Gestaltung (Entwicklung) sowie die Pflege (Weiterentwicklung) von betrieblichen Informationssystemen. Eine Voraussetzung für die Erfüllung dieser Zielsetzung ist die Verknüpfung von Bestandteilen des SOM-Geschäftsprozessmodells mit den Komponenten der RESTful SOA im Rahmen der Modellierung von IS. Damit einher geht die Bildung einer gemeinsamen methodischen Basis und einem einheitlichen Begriffssystem für die modellbasierte Überführung von Struktur- und Verhaltenseigenschaften der Geschäftsprozessmodelle in die Spezifikation der RESTful SOA (erstes Teilziel). Auf dieser Basis erfolgt die Konstruktion eines Lösungsansatzes zur durchgängigen Pflege von betrieblichen Informationssystemen mit dem Ziel, deren Modellebenen an Veränderungen in SOM-Geschäftsprozessmodellen anzupassen (zweites Teilziel).

FORMALZIELE

Die Entwicklungsmethodik stellt den Lösungsansatz für das Untersuchungsproblem der Arbeit dar. Folgende *Formalziele* werden mit der Konstruktion der Methodik verfolgt:

- Das betriebliche Informationssystem und die durchzuführende Systementwicklungsaufgabe weisen eine hohe *Komplexität* auf. Die zu konstruierende Methodik soll Hilfsmittel bereitstellen, die geeignet sind, den Systementwickler bei der *Bewältigung* dieser Komplexität zu unterstützen.
- Durch die Methodik soll die Sicherstellung von *Konsistenz* auf den Modellebenen des spezifizierten Informationssystems sowie den Beziehungen zwischen diesen gefördert werden. Die Ausrichtung von Komponenten der RESTful SOA auf die Bestandteile des SOM-Geschäftsprozessmodells sowie die bestehenden Abhängigkeiten zwischen fachlichen Anforderungen und ihrer Umsetzung wären dann lückenlos prüfbar und damit nachvollziehbar.
- Der erarbeitete Lösungsansatz soll unter Einhaltung der Restriktionen des Untersuchungsobjekts an problemspezifische Anforderungen der Systementwicklungsaufgabe *flexibel anpassbar* sein (z. B. eine Veränderung der eingesetzten Technologie bzw. Plattform der Zielarchitektur). Mit der Konstruktion einer flexiblen Methodik wird die Bewältigung von Änderungen an der Systementwicklungsaufgabe selbst ermöglicht. Ergänzend dazu wird die *Unabhängigkeit* der spezifizierten RESTful SOA von konkreten Implementierungstechnologien oder Plattformen gefordert, so dass die Realisierung auf mehreren Basismaschinen erfolgen kann.

Der Lösungsansatz zur Behandlung des Untersuchungsproblems der Arbeit ist dem Bereich der konzeptuellen Modellierung von Informationssystemen zuzuordnen [Stra13a]. Mit der Modellbildung wird generell das Ziel der Aufstellung „guter“ Modelle verfolgt. Der Lösungsansatz dieser Arbeit soll den Modellierer demnach bei der Aufstellung „umfassender“, „richtiger“ und „geeigneter“ Modelle auf den Ebenen der Systementwicklung unterstützen [Sinz98]. Die erstellten Modelle sind also ein struktur- und verhaltenstreues sowie zweckorientiertes Abbild

des betrachteten betrieblichen Informationssystems, und konsistent und vollständig zum definierten Begriffssystem [FeSi13, S. 134 ff.].

Lösungsstrategie

Zur Erreichung des Untersuchungsziels wird ein schrittweises Vorgehen gewählt. Zunächst werden die methodischen und softwaretechnischen Grundlagen des Untersuchungsgegenstandes als Voraussetzung für die Methodenkonstruktion eingeführt. Aufgrund der Komplexität des Untersuchungsproblems wird die Entwicklungsmethodik in zwei Schritten konstruiert, die jeweils mit dem Erreichen eines Teilziels der Arbeit korrespondieren. Im ersten Konstruktionsschritt wird ein einheitliches Begriffssystem zur konzeptuellen Modellierung des Untersuchungsobjekts erarbeitet. Darauf aufbauend wird der Einsatz methodischer Hilfsmittel bei der Durchführung einer modellbasierten Entwicklung von RESTful SOA ausgehend von SOM-Geschäftsprozessmodellen behandelt. Auf Basis dieser Ergebnisse erfolgt im zweiten Schritt die Erweiterung der konstruierten Methodik um einen Lösungsansatz zur Unterstützung der zielgerichteten Anpassung eines bestehenden Modellsystems an Veränderungen, deren Ursprung in den strukturflexiblen Geschäftsprozessen liegt. Dabei sind die nicht-automatisierbaren, personell durchzuführenden Entwicklungsaufgaben von den automatisierbaren Aufgaben abzugrenzen und dem Systementwickler methodische Hilfsmittel zur Unterstützung ihrer Durchführung bereitzustellen.

Fallstudie und Werkzeugprototyp

Der Proof of Concept und Nutzen des konzipierten Lösungsansatzes wird anhand einer *Fallstudie* und eines *prototypischen Entwicklungswerkzeugs* nachgewiesen. Ausgehend vom SOM-Geschäftsprozessmodell der Fallstudie werden die Konzepte der Entwicklungsmethodik zur modellbasierten Gestaltung sowie der modellbasierten Pflege von RESTful SOA durch ihre Anwendung auf einen konkreten Sachverhalt beispielhaft erläutert. Die Realisierbarkeit einer Werkzeugunterstützung für die einzelnen Lösungsansätze der Methodik wird anhand ihrer prototypischen Implementierung auf Basis einer Metamodellierungsplattform demonstriert.

Akteure der Methodennutzung

Die Akteure einer Methodennutzung lassen sich anhand der beteiligten Subjekte der Modellnutzung und -erstellung auf den Abstraktionsebenen der Entwicklung differenzieren (z. B. [Stra13c]). Auf der Ebene der Geschäftsprozesse dienen Modelle als Mittel zur Kommunikation mit dem interessierten Nutzer. Zudem erfolgt die Gestaltung und Pflege von (SOM-) Geschäftsprozessmodellen durch die Domänenexperten des betrieblichen Systems [FrLa03, S. 14]. Die Überführung der Modelle von der Geschäftsprozess- auf die softwaretechnische Ebene kann durch den Einsatz interdisziplinär geschulter Modellierer (z. B. Wirtschaftsinformatiker) unterstützt werden. Auf Ebene der Softwarearchitektur erfolgt die Spezifikation der RESTful SOA durch den Systementwickler, der überlappend mit der Rolle des Modellierers definiert sein kann. Die methodische Überbrückung der semantischen Lücke ist eine wichtige Voraussetzung für die Kommunikation zwischen den beteiligten Nutzern und zur Prüfung von Beziehungen zwischen Modellen unterschiedlicher Abstraktionsebenen [Fran98, S. 6]. Die konstruierte Methodik erlaubt den Nutzern die Betrachtung und Analyse des modellierten Informationssystems aus unterschiedlichen Perspektiven.

ANMERKUNG ZUR FORSCHUNGSMETHODIK

Mit der vorliegenden Arbeit wird die Lösung eines *Konstruktionsproblems* angestrebt, das die Gestaltung und Pflege von RESTful SOA auf Basis von flexiblen SOM-Geschäftsprozessmodellen zum Gegenstand hat. Ergebnis der Untersuchung ist die Konstruktion eines Artefakts in Form einer Entwicklungsmethode ([MaSm95], [Fers79, S. 43 ff.]). Durch die Methode werden Handlungsanleitungen und Hilfsmittel zur Konstruktion und zum Betrieb von Informationssystemen für den aufgespannten Untersuchungsbereich bereitgestellt ([ÖWB10, S. 3], [Grei03, S. 956 f.]). Nach [WiHe07], [WiHe06] lässt sich der Forschungsansatz als konzeptionell-deduktive Analyse klassifizieren. Die Arbeit ist der gestaltungsorientierten IS-Forschung zuordenbar [ÖWB10].

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in insgesamt neun Kapitel. Jedem Kapitel wird eine Einführung vorangestellt, welche seine inhaltlichen Ausführungen überblicksartig darstellt und in den Gesamtzusammenhang der Arbeit rückt. Die zentralen Ergebnisse werden am Kapitelende zusammengefasst.
Die Einleitung und Abgrenzung der Untersuchungssituation dieser Arbeit erfolgt in *Kapitel 1*.

Die Grundlagen und zentralen Begriffe der Modellierung betrieblicher Informationssysteme, der Systementwicklung sowie der modellgetriebenen Entwicklung von Softwaresystemen werden in *Kapitel 2* eingeführt und erläutert. Sie bilden die methodischen Grundlagen für die Konstruktion der Entwicklungsmethodik.

Die Erläuterung der Grundlagen von RESTful SOA sind Gegenstand von *Kapitel 3*. Aufbauend auf einer Einführung von Merkmalen, Bestandteilen und grundlegendem Aufbau serviceorientierter Architekturen sowie von REST als Architekturstil zur Realisierung von verteilten Systemen, erfolgt die Konzeption von RESTful SOA als Realisierungsform verteilter AwS.

Den Schwerpunkt von *Kapitel 4* bildet die Untersuchung flexibler SOM-Geschäftsprozessmodelle, welche den Startpunkt des Systementwicklungsprozesses in dieser Arbeit darstellen. Hierzu werden die Grundlagen der Modellierung von Geschäftsprozessen mit der SOM-Methodik sowie eine tiefergehende Untersuchung von Flexibilität in betrieblichen Systemen durchgeführt. Abschließend wird eine Fallstudie eingeführt, anhand derer die konzeptionellen Ausführungen in den weiteren Kapiteln exemplarisch erläutert werden.

Kapitel 5 behandelt mit der Konstruktion der Entwicklungsmethodik zur modellbasierten Gestaltung von RESTful SOA auf Basis von SOM-Geschäftsprozessmodellen den Lösungsansatz zur Erfüllung des ersten Untersuchungsziels. Im Rahmen einer Erweiterung der SOM-Methodik werden Architekturmodell, Vorgehensmodell, ein Modellierungsansatz für RESTful SOA sowie die Entwicklungsschritte zur methodisch geleiteten Bildung der Modellebenen des zu spezifizierenden Systems konzipiert.

Gegenstand von *Kapitel 6* ist die Erweiterung der im fünften Kapitel konstruierten Entwicklungsmethodik um einen Lösungsansatz zum Erreichen des zweiten Untersuchungsziels. Die modellbasierte Anpassung von RESTful SOA an flexible SOM-Geschäftsprozessmodelle

unterstützen ein Dokumentations-, Modellvergleichs- und Empfehlungsansatz sowie ein Vorgehensmodell als Bestandteile der erweiterten Methodik zur modellbasierten Systempflege.

In *Kapitel 7* erfolgt die Konzeption und prototypische Implementierung einer Werkzeugunterstützung für die konstruierte Methodik. Zunächst werden die gewählte Entwicklungsplattform sowie die Technologien des Prototyps vorgestellt und die methodischen Rahmenbedingungen einer Realisierung untersucht. Anschließend werden die Komponenten des Prototyps erläutert und ihr Einsatz beispielhaft gezeigt.

Verwandte Arbeiten aus den Themenbereichen der Arbeit werden in *Kapitel 8* beleuchtet. Hierzu wird zunächst der Stand der Literatur für den Untersuchungsgegenstand RESTful SOA vorgestellt und anschließend die konstruierte Methodik ausgewählten Ansätzen zur modellgetriebenen Entwicklung von SOA gegenübergestellt. Des Weiteren werden Konzepte zur modellgetriebenen Anpassung von Softwaresystemen betrachtet.

Die Ergebnisse der Arbeit werden in *Kapitel 9* kritisch gewürdigt. Abschließend wird ein Ausblick auf mögliche Forschungsfelder im Themengebiet der Arbeit gegeben.

KONVENTIONEN

Bedeutsame Textstellen werden *kursiv* hervorgehoben. Die Betonung ordnender Begriffe erfolgt mit der Formatierung **fett.** Autorennamen sind durch KAPITÄLCHEN und Codebeispiele durch den Einsatz `nichtproportionaler Schrift` gekennzeichnet. Wörtlich übernommene Textstellen fremder Quellen werden in Anführungszeichen gesetzt und „*kursiv*“ formatiert. Im laufenden Text werden Literaturquellen in Form von Kurzbelegen nach dem Schema [<Autorenkürzel><Jahr>, S. <Seitenzahl>] angegeben.

2 Grundlagen der modellbasierten Systementwicklung

Das zweite Kapitel gibt eine Einführung in Konzepte und Methoden der modellbasierten Entwicklung von Systemen. Zunächst werden die grundlegenden Begriffe und Hilfsmittel für die Modellierung betrieblicher Informationssysteme erläutert. Anschließend erfolgt eine allgemeine Einführung in die Systementwicklung. Eine Betrachtung des Einsatzes von Modellen als zentrale Artefakte zur modellbasierten Entwicklung von Systemen sowie eine thematische Einordnung des Ansatzes der vorliegenden Arbeit schließen die Ausführungen.

2.1 Methodische Grundlagen der Modellierung betrieblicher Informationssysteme

Der zentrale Forschungsgegenstand der Wirtschaftsinformatik sind die Informationssysteme von Organisationen in Wirtschaft und Verwaltung (z. B. [ÖWB10, S. 3 f.], [FeSi13, S. 3]). Im nachfolgenden Abschnitt findet eine Charakterisierung und Abgrenzung dieses Betrachtungsgegenstandes statt (Abschnitte 2.1.1 und 2.1.2). Betriebliche Systeme, Informationssysteme und Anwendungssysteme weisen im Allgemeinen eine hohe Komplexität auf. Modelle sind das wichtigste Hilfsmittel für die Analyse, Gestaltung und Dokumentation von komplexen Systemen sowie die Kommunikation zwischen den Modellierern und/oder Modellnutzern (Abschnitte 2.1.3 und 2.1.4). Zusätzlich unterstützen Metaphern, Metamodelle und Modellarchitekturen die Bildung und Nutzung von Modellen durch das Aufspannen eines methodischen Beschreibungsrahmens (Abschnitte 2.1.5 und 2.1.6).

2.1.1 Betriebliches System

Ein *betriebliches System* stellt aus systemorientierter Sicht ein offenes, zielgerichtetes und sozio-technisches System dar. Ein Betrieb interagiert mit seiner Umwelt durch den Austausch von Leistungen oder von Nachrichten zur Lenkung der Leistungen (*offen*), richtet sein Verhalten auf ein Zielsystem aus (*zielgerichtet*) und setzt zur Durchführung seiner Aufgaben Menschen und Maschinen ein (*sozio-technisch*) [FeSi13, S. 65 f.]. Beispiele für betriebliche Systeme sind Unternehmen, Unternehmensverbünde oder auch Teilbereiche von Unternehmen. Diese erstrecken sich über die unterschiedlichen Domänen in Wirtschaft und Verwaltung, wie z. B. Industrie, Handel, Dienstleistung oder öffentliche Betriebe und Verwaltungen [FeSi13, S. 3].

Das Verhalten betrieblicher Systeme ist auf die Erfüllung von Zielsystemen ausgerichtet [FeSi13, S. 65]. Art und Zweck der betrieblichen Leistungserstellung wird dabei in Form von *Sachzielen* beschrieben. Die Qualität der Leistungserstellung und damit der Sachzielerreichung legen die zugehörigen *Formalziele* aus wirtschaftlicher, z. B. Gewinnmaximierung, und aus technischer Sicht, z. B. Qualitätsmerkmale eines Produktes oder des Prozesses, fest (z. B. [FeSi13, S. 72], [SCS99, S. 416 f.]). Die Abläufe zur Erstellung und Übergabe von Leistungen oder deren Koordination werden als *betriebliche Prozesse* bezeichnet (z. B. [Sinz13c, S. 5]).
Ein Beispiel hierfür ist der Beschaffungsprozess eines Produktionsbetriebes. Von einem externen Lieferanten werden Produktionsgüter beschafft und der Fertigung als internen

Teilbereich des Betriebes bereitgestellt. Die Leistung des Prozesses besteht in der Beschaffung und Bereitstellung bestimmter Güter. Die Koordination der Leistungsübergabe zwischen Lieferant und Produktionsbetrieb erfolgt durch den Austausch von Informationen. Die betriebliche Leistungserstellung und deren Koordination werden auf der Aufgabenebene in Form von *Geschäftsprozessen* spezifiziert (Abschnitt 4.1).

Voraussetzung für die Untersuchung betrieblicher Systeme ist die zweckorientierte Abgrenzung des relevanten Ausschnitts der betrachteten betrieblichen Realität, der als betriebliche *Diskurswelt* bezeichnet wird. Weiter sind für eine Untersuchung diejenigen Teile der betrieblichen *Umwelt* interessant, die mit der Diskurswelt in Beziehung stehen. Die Diskurswelt und der relevante Ausschnitt der Umwelt bilden das betriebliche *Objektsystem* [FeSi13, S. 7].

2.1.2 Betriebliches Informationssystem und Anwendungssystem

FERSTL und SINZ verstehen unter einem *betrieblichen Informationssystem* (*IS*) dasjenige Teilsystem eines betrieblichen Systems, „*das Informationen verarbeitet, d.h. erfasst, überträgt, transformiert, speichert und bereitstellt*“ [FeSi13, S. 3].
Der IS-Begriff wird somit umfassend interpretiert und dient der Erfüllung der Aufgaben „Lenkung der betrieblichen Leistungserstellung“ sowie „Erstellung von informationsbasierten Dienstleistungen“. Jedes IS ist damit Teil eines Systems, welches sich anhand geeigneter Abgrenzungskriterien wiederum in weitere, teils überlappende Teilsysteme untergliedern lässt [FeSi13, S. 6 ff.]. Das Ergebnis der Abgrenzung fasst Abbildung 2.1 zusammen.

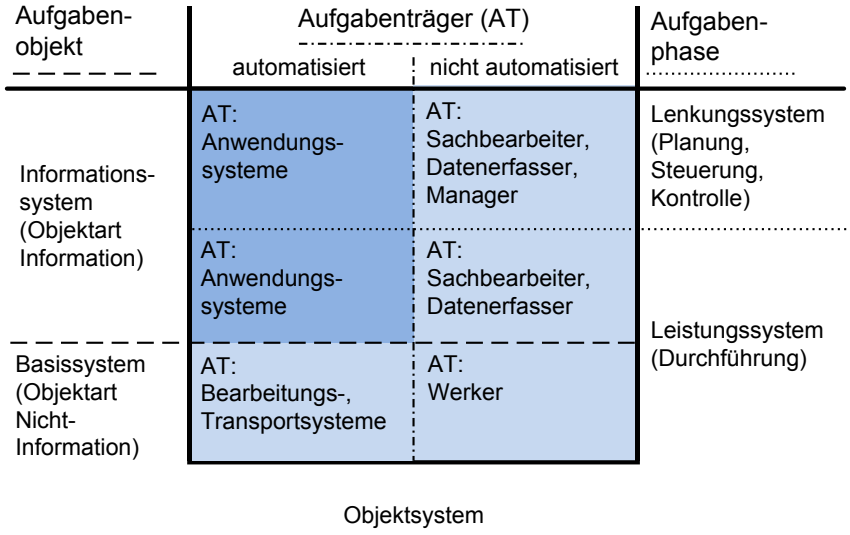


Abbildung 2.1: Abgrenzung von Teilsystemen des betrieblichen Objektsystems [FeSi13, S. 6]

Als Beziehungsarten zwischen Objekten werden *Informationen* oder *Nicht-Informationen* unterschieden (Abgrenzungskriterium *Objektprinzip*), die zur Bildung der Teilsysteme betriebliches *Informationssystem* sowie *Basissystem* führen. Das Basissystem verarbeitet Nicht-Informationen, also Güterflüsse, Zahlungsflüsse, Energieflüsse oder reale Dienstleistungen. Anhand der Phasen Planung, Steuerung und Kontrolle der Leistungserstellung bzw. der Durchführung der Leistungserstellung wird das *Lenkungssystem* bzw. das *Leistungssystem* des Objektsystems bestimmt (Abgrenzungskriterium *Aufgabenphase*). Die Differenzierung nach dem Abgrenzungskriterium *Aufgabenträger* (AT) führt zu einem automatisierten und nicht-

automatisierten Teilsystem. *Anwendungssysteme* sind maschinelle Aufgabenträger des IS und führen die Informationsverarbeitung automatisiert durch. Im Basissystem kommen z. B. Bearbeitungs- und Transportsysteme zum Einsatz. Das nicht-automatisierte Teilsystem von Informations- und Basissystem umfasst den *personellen Aufgabenträger* Mensch [FeSi13, S. 7].

2.1.3 Strukturmodell der betrieblichen Aufgabe

Grundlage für die Analyse und Gestaltung der Aufgabenebene in betrieblichen Informationssystemen ist das Konzept der betrieblichen Aufgabe. Den Begriff der Aufgabe definiert KOSIOL als *„Zielsetzungen für zweckbezogene menschliche Handlungen“* [Kosi76, S. 43]. Demnach erfolgt die zielorientierte Verrichtung einer Aufgabe an einem Gegenstand (*Aufgabenobjekt*) unter Verwendung sachlicher *Arbeits-* oder *Hilfsmittel*. Die Verrichtung selbst vollzieht sich innerhalb eines bestimmten *Raumes* und einer bestimmten *Zeitspanne* ([Kosi76, S. 43 ff.], [FeSi13, S. 97 f.]). FERSTL und SINZ stellen ausgehend von diesem Verständnis ein Strukturmodell von Aufgaben vor, welches zwischen einer Außen- und Innensicht unterscheidet [FeSi13, S. 98 ff.]. Das Modell zeigt Abbildung 2.2.

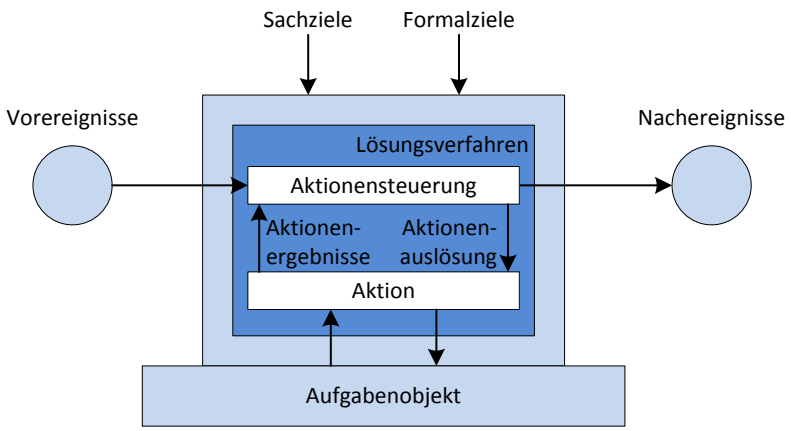


Abbildung 2.2: Modell der Aufgabenstruktur [FeSi13, S. 98]

- Eine Aufgabe weist aus *Außensicht* (hellblau) die Merkmale Aufgabenobjekt, Sach- und Formalziele, Vorereignisse sowie Nachereignisse auf. Während *Vorereignisse* eine Aufgabendurchführung auslösen, führt die Durchführung der Aufgabe zu *Nachereignissen*. Das *Aufgabenobjekt* ist der Gegenstand der Verrichtung. Es besitzt einen Zustandsspeicher auf den über bereitgestellte Input- und Output-Kanäle manipulierend zugegriffen wird [FeSi13, S. 98].

- Die *Innensicht* einer Aufgabe (dunkelblau) spezifiziert das *Lösungsverfahren* (Verrichtungsvorgang) zur Erreichung der Sachziele unter Berücksichtigung der Formalziele. Es umfasst eine Menge von *Aktionen*, deren Reihenfolge durch eine *Aktionssteuerung* bestimmt wird. Aktionen wirken sequentiell oder parallel auf das Aufgabenobjekt ein oder erfassen dessen Zustände. Ist eine Aktion funktional beschreibbar, so ist sie automatisierbar und kann von maschinellen Aufgabenträgern (z. B. Anwendungssystem) ausgeführt werden. Ist sie nicht funktional beschreibbar, so wird sie personell durchgeführt und ist somit nicht-automatisierbar. Das Lösungsverfahren wird demnach stets in Abhängigkeit vom *Aufgabenträgertyp* (Mensch, Maschine) beschrieben [FeSi13, S. 98].

Die Durchführung einer Aufgabe bzw. ihres Lösungsverfahrens durch einen Aufgabenträger wird als *Vorgang* bezeichnet [FeSi13, S. 99]. Hierbei wird zwischen dem *Vorgangstyp* und den *Vorgangsinstanzen* (konkreter Vorgang) unterschieden. Während der Vorgangstyp die Durchführung des Lösungsverfahrens auf einem zugehörigen Aufgabenobjekt festlegt, beschreibt eine Vorgangsinstanz die eigentliche Aufgabendurchführung unter Nutzung eines konkreten Aufgabenträgers (Aufgabenträgerinstanz). Das Zusammenwirken von Vorgängen (einzelner Teilaufgaben) zur Realisierung des Lösungsverfahrens der (Gesamt)-Aufgabe wird in einem *Vorgangsnetz* definiert [FeSi13, S. 62].

In Abhängigkeit von dem Zugriff auf die Aufgabenobjekte können die Beziehungen zwischen Aufgaben als enge oder lose *Kopplung* realisiert werden. Bearbeiten mehrere Aufgaben ein gemeinsames Aufgabenobjekt, so sind diese eng gekoppelt. Eine lose Kopplung zwischen Aufgaben besteht, wenn diese getrennte Aufgabenobjekte besitzen und Informationen über Nachrichtenkanäle austauschen [FeSi13, S. 101].

Durch Zuordnung von Aufgabenträgern zu informationsverarbeitenden Aufgaben wird deren *Automatisierungsgrad* bestimmt [FeSi13, S. 55]. Demnach ist eine Aufgabe *vollautomatisiert* (*automatisiert*), wenn ihre Durchführung vollständig durch den Einsatz maschineller Aufgabenträger erfolgt. Wird eine Aufgabe gemeinsam von personellen und maschinellen Aufgabenträgern durchgeführt, so ist sie *teilautomatisiert*. *Nicht-automatisierte* Aufgaben werden ausschließlich von personellen Aufgabenträgern durchgeführt.

2.1.4 Modellbegriff und Modellierung

Modelle stellen in der Wirtschaftsinformatik das wichtigste Hilfsmittel zur Analyse, Gestaltung und Pflege betrieblicher Informationssysteme dar (z. B. [FSF+14, S. 49 ff.], [FeSi13, S. 133], [Stra13c], [WiHe07, S. 282], [Thom05, S. 5] u. a.). Als zweckgerichtet konstruierte Abstraktion des Betrachtungsgegenstandes *Informationssystem* unterstützen Modelle zum einen die Beherrschung dessen Komplexität, zum anderen sind sie ein geeignetes Medium zur Analyse, Gestaltung und Dokumentation von IS sowie zur Kommunikation zwischen beteiligten Subjekten ([FSF+14, S. 49], [FeSi13, S. 135 f.]). STACHOWIAK betont Abbildung, Verkürzung und Pragmatik als die drei Hauptmerkmale des allgemeinen und zweckorientierten Modellbegriffs [Stac73, S. 131 ff.].

Als *Modellierung* wird im Folgenden die „*methodisch geleitete Tätigkeit der Erstellung von Modellen*“ verstanden [FeSi13, S. 133]. Da IS im Wesentlichen sprachliche Konzepte, und keine physischen Objekte beschreiben, erfolgt ihre Erfassung meist in Form *konzeptueller Modelle* ([BePf06], [Stra13a]). Die an der Modellierung beteiligten Subjekte sind der *Modellierer* (synonym: Modellersteller oder Modellkonstrukteur) sowie der *Modellnutzer*. Modellierer und Modellnutzer können, müssen aber nicht übereinstimmen ([Stra13c], [FeSi13, S. 135 f.]).

FERSTL und SINZ geben eine *informale Definition* des Begriffs Modell als „*ein System*, *das ein anderes System zielorientiert abbildet*“ [FeSi13, S. 22]. Die Erstellung eines Modells ist damit stets zielorientiert und demzufolge zweckorientiert bezüglich der Abbildung des untersuchten Systems.
Die Autoren definieren ein Modell M *formal* als *3-Tupel* (S_O, S_M, *f*), wobei S_O das Objektsystem

(das abzubildende System), S_M das Modellsystem (das Abbild des Objektsystems) und f die Modellabbildung (Abbildungsbeziehung) zwischen den Komponenten V_O und V_M der Systeme S_O und S_M beschreibt [FeSi13, S. 22 f.]. Über die Modellabbildung f: $V_O \rightarrow V_M$ werden die Komponenten V_O des zu modellierenden Objektsystems S_O auf die Komponenten V_M des zu erstellenden Modellsystems S_M abgebildet (Abbildung 2.3). *Strukturtreue* und *Verhaltenstreue* sind hierbei die wesentlichen Anforderungen bezüglich einer Abbildung von S_O auf S_M. Sie werden durch eine homomorphe (bzw. isomorphe) Modellabbildung f erreicht [FeSi13, S. 134].

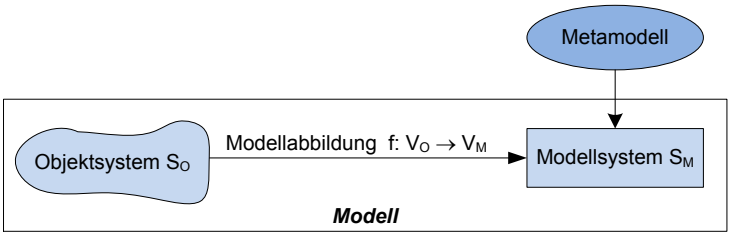


Abbildung 2.3: Modell und Metamodell [FeSi13, S. 135]

In der Wirtschaftsinformatik stellen Objektsysteme meist einen Ausschnitt der betrieblichen Realität und damit ein reales System dar [Sinz95, S. 1]. Das Modellsystem ist normalerweise ein immaterielles System und wird als formales System spezifiziert. Zur Modellierung von komplexen Systemen sind insbesondere semi-formale Ansätze geeignet, die einen informalen Anteil besitzen und ihre Semantik, aber auch Teile ihrer Syntax nicht-formal definieren ([FeSi13, S. 143], [Sinz95, S. 1]). Semi-formale Modellsysteme werden z. B. in graphischer Form unter Verwendung einer semi-formalen Modellierungssprache dargestellt (z. B. [FeSi13, S. 143 ff.], [HGN10, S. 31], [Schl04, S. 59], [FrLa03]). Eine methodische Unterstützung der Spezifikation des Modellsystems erfolgt auf Basis eines Metamodells, das verfügbare Modellbausteine sowie deren Beziehungen, Verknüpfungsregeln und die zugehörige Semantik definiert (Abschnitt 2.1.5). Das gebildete Modellsystem selbst muss formal *konsistent* und *vollständig* in Bezug auf das Metamodell sein, also dessen Gesetzmäßigkeiten genügen [FeSi13c, Kapitel 2, S. 21]. Der Zusammenhang zwischen Modell und Metamodell ist in Abbildung 2.3 dargestellt.

KONSTRUKTIVISTISCHES MODELLIERUNGSVERSTÄNDNIS

Der Modellierer nimmt bei der Bildung des Modells eine zentrale Rolle ein. Für eine nähere Betrachtung wird hierzu dem bisher angenommenen abbildungsorientierten Modellverständnis das konstruktivistische Modellierungsverständnis gegenübergestellt (z. B. [FeSi13, S. 135 f.], [Stra13c], [Thom05, S. 13 ff.]).

Nach dem *konstruktivistischen Modellierungsverständnis* wird stets zielorientiert modelliert (Abbildung 2.4). Dem Subjekt sind die Modellziele bekannt und es richtet seine Modellierungstätigkeit auf deren Erreichen aus. Hierzu erfolgt eine Abgrenzung und Interpretation der betrieblichen Realität. Da der Modellierer als Subjekt jedoch mit der betrieblichen Realität in einer Kontextbeziehung steht, unterliegt die Perzeption (Erfassung) der Realität stets subjektiven Einflüssen [FeSi13, S. 135]. Es werden somit geeignete Konzepte benötigt, um die subjektiven Einflüsse auf die Modellbildung und Modellnutzung zu reduzieren oder zumindest zu begrenzen. Wichtige Hilfsmittel dafür sind Metaphern und Metamodelle.

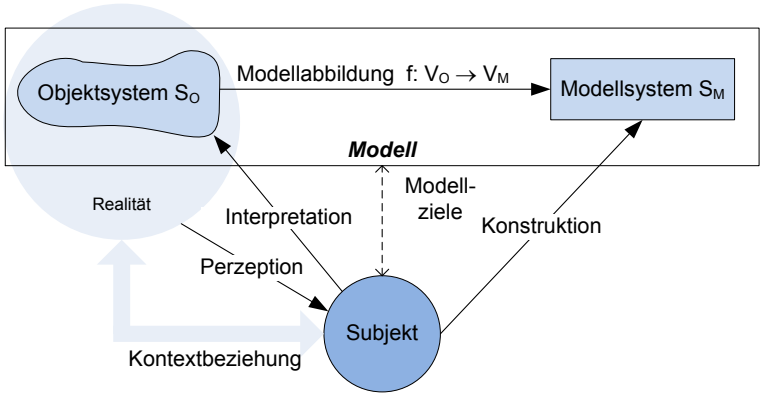


Abbildung 2.4: Konstruktivistisches Modellierungsverständnis [FeSi13, S. 136]

Den weiteren Ausführungen dieser Arbeit wird der Modellbegriff nach FERSTL und SINZ zugrunde gelegt. Für eine tiefergehende Betrachtung des Verständnisses, Einsatzes, Nutzens oder der Evaluation von Modellen in der Wirtschaftsinformatik sei auf die weiterführende Literatur verwiesen (z. B. [Stra13c], [Thom05], [Fett09], [Fran98]).

2.1.5 Metapher, Metamodell und Meta-Metamodell

Wie im vorangegangenen Abschnitt motiviert, werden Hilfsmittel benötigt, die den Modellierer bei der Bewältigung der zum Teil hohen Komplexität von Modellsystemen unterstützen. Wichtige Konzepte für die Erfassung, Interpretation und Konstruktion von Systemen sind die der Metapher und des Metamodells ([FeSi13, S. 136 ff.], [Stra13b]).

Der Begriff *Metapher* wird allgemein als sprachlicher Ausdruck verstanden,

> „*der eine aus einem bestimmten Bedeutungszusammenhang stammende, bildhafte Vorstellung auf einen anderen Bedeutungszusammenhang anwendet*“ [FeSi13, S. 136].

In der Modellierung unterstützen Metaphern den Modellierer bei der Einnahme einer bestimmten Perspektive, auf deren Basis die Perzeption und Interpretation des Objektsystems sowie die Konstruktion des Modellsystems erfolgt [FeSi13, S. 136]. Die Übertragung dieser Analogie aus der realen Welt erleichtert zudem eine Interpretation des Modellsystems durch den Modellnutzer.

Als *Metamodell* wird allgemein ein Beschreibungsmodell für die Sprache verstanden, mit der ein Modellsystem formuliert wird [Stra13b]. Nach FERSTL und SINZ definiert ein Metamodell

> „*die verfügbaren Arten von Modellbausteinen, die Arten von Beziehungen zwischen Modellbausteinen, die Regeln für die Verknüpfung von Modellbausteinen durch Beziehungen sowie die Bedeutung (Semantik) der Modellbausteine und Beziehungen*“ [FeSi13, S. 137].

Das Metamodell legt somit das Begriffssystem der Modellierungssprache fest, mit dem das Modellsystem spezifiziert wird. Dem Begriffssystem des Metamodells liegt eine Metapher zugrunde, auf die es abgestimmt ist [FeSi13, S. 137].
Die Festlegung von Metamodell und Metapher spezifiziert einen *Modellierungsansatz*. Dieser beschreibt einen Gestaltungsrahmen zur Unterstützung der beteiligten Subjekte bei der Sicht auf Objekt- und Modellsystem sowie der Spezifikation des Modellsystems [FeSi13, S. 136].

Metamodelle werden auf Basis eines *Meta-Metamodells* formuliert. Dieses spezifiziert das hierfür einheitliche Begriffssystem und damit die benötigte Beschreibungssprache [Stra13b]. In der vorliegenden Arbeit wird das Meta-Metamodell nach SINZ zur Spezifikation von Metamodellen verwendet ([Sinz96, S. 129], [FeSi13, S. 138 f.]). Seinen Aufbau zeigt Abbildung 2.5.

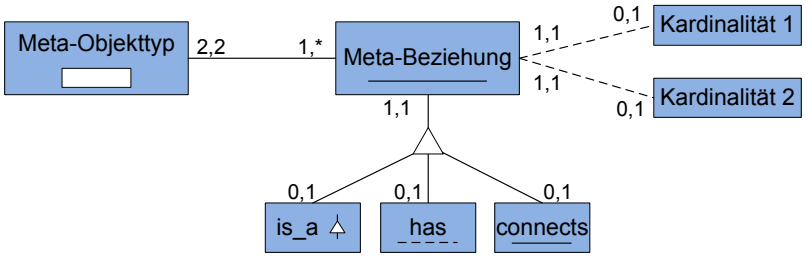


Abbildung 2.5: Meta-Metamodell (in Anlehnung an [Sinz96, S. 129])

Das Meta-Metamodell definiert als Bausteine den *Meta-Objekttyp* und die *Meta-Beziehung*. Eine Meta-Beziehung verbindet genau zwei Meta-Objekttypen und kann durch zwei Kardinalitäten in der (min,max)-Notation präzisiert werden. Eine Meta-Beziehung ist entweder eine Generalisierungs- (*is_a*), Attributzuordnungs- (*has*) oder Assoziationsbeziehung (*connects*) [FeSi13, S. 138 f.]. Da sich das Meta-Metamodell selbst beschreibt, wird kein weiterer Beschreibungsrahmen benötigt.

Als *Schema* wird in diesem Zusammenhang ein nach einem Modellierungsansatz erstelltes Modell bezeichnet (z. B. [FeSi13, S. 138]). Zu jedem Schema können beliebig viele *Ausprägungen* (*Instanzen)* gebildet werden. Meta-Metamodell, Metamodell, Schema und Ausprägung bilden jeweils eine *Metaebene* der Modellierung: Meta-Metaebene (3), Metaebene (2), Schemaebene (1) sowie Ausprägungsebene (0). Jede Metaebene (i) stellt dabei eine *Extension* der jeweils nächst-höheren Metaebene (i+1) dar [FeSi13, S. 138]. Ein Metamodell besteht somit aus *Metaobjekten* (Objekttypen des Modells), die durch Beziehungen verknüpft sind. Auf der Schemaebene wird das Modell in Form von *Schemaobjekten* (Modellobjekten) und den Beziehungen zwischen diesen beschrieben.

Ein Datenschema besteht beispielsweise aus den zwei Datenobjekttypen (Metaobjekte) *Kunde* und *Auftrag* sowie der Beziehung zwischen beiden Objekten. Als Extension der Schemaebene umfasst die Ausprägungsebene Instanzen der Datenobjekttypen, z. B. Schmidt und Müller als konkrete Kunden. Die Metaebene besteht aus dem Metamodell (Datenmodell) zur Modellierung des Datenschemas (vgl. [FeSi13, S. 139]).

Hinweis: Der Begriff *Baustein* (eines Modells bzw. Schemas) bezeichnet meist die modellierten Objekte, die als Metaobjekte im zugrundeliegenden Metamodell definiert sind. Zusammen mit den Beziehungen zwischen Bausteinen (Meta-Beziehung des Metamodells) werden sie in den weiteren Ausführungen auch als *Modellelemente* bezeichnet.

2.1.6 Generischer Architekturrahmen

Architekturen sind in der Wirtschaftsinformatik ein wichtiges Hilfsmittel zur Unterstützung der Analyse und Gestaltung von Informationssystemen (z. B. [WiAi13], [Sinz13a], [Sche01], [Wall96, S. 26 ff.]). Als grundlegende Metapher wird bei der Architekturgestaltung meist die

Sicht auf das Informationssystem als Bauwerk genutzt ([Stei93, S. 9 f.], [FeSi13, S. 195]). Entsprechend diesem Verständnis nehmen Entwickler von Informationssystemen bzw. von Modellsystemen des IS die Rolle eines Architekten ein. Nach FERSTL und SINZ umfasst eine *Architektur*

- *„den Bauplan eines Objektsystems im Sinne einer Spezifikation seiner Komponenten und ihrer Beziehungen unter allen relevanten Blickwinkeln sowie*
- *die Konstruktionsregeln für die Erstellung des Bauplans*“ [FeSi13, S. 195].

Der Bauplan eines Objektsystems ist das gebildete Modellsystem. Die Konstruktionsregeln für die Erstellung des Bauplans beschreibt das zugehörige Metamodell [FeSi13, S. 195].

Als letztes methodisches Hilfsmittel zur Unterstützung der IS-Modellierungsaufgabe wird nachfolgend der *generische Architekturrahmen* vorgestellt, der durch die zielgerichtete Strukturierung eines Modellsystems dessen Komplexität beherrschbar macht ([Sinz95, S. 2 f.], [Sinz97, S. 3]). Abbildung 2.6 zeigt den Aufbau des generischen Architekturrahmens, der auf der Metaebene (2) beschrieben ist. Konkrete Architekturen sind somit Extensionen des Architekturrahmens auf Schemaebene (1).

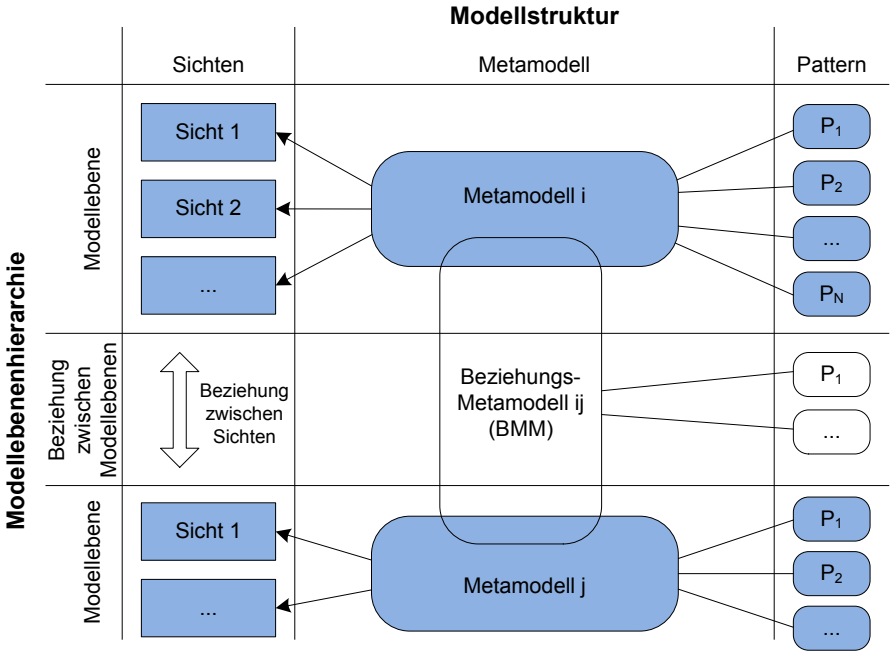


Abbildung 2.6: Generischer Architekturrahmen (in Anlehnung an [Sinz97])

Ein umfassendes Modellsystems wird zielorientiert in *Modellebenen* untergliedert, die dieses unter einer bestimmten Perspektive beschreiben. Zur Spezifikation der Modellebene definiert das Metamodell ein Begriffssystem. Zudem dienen Sichten der Bewältigung der Komplexität innerhalb einer *Modellebene* und beschreiben diese anhand bestimmter Systemmerkmale. Sichten werden als Projektionen auf ein Metamodell definiert. Zur Unterstützung der Modell-

bildung einer Modellebene werden darüber hinaus *Pattern* (P_1… P_N) (Strukturmuster[1]) angegeben, die Lösungen oder Lösungsverfahren für den Entwurf eines Teilmodellsystems dokumentieren. Metamodell, Sichten und Pattern beschreiben die *Modellstruktur* einer Modellebene [Sinz97, S. 3 f.].

Die *Modellebenenhierarchie* setzt die einzelnen Modellebenen innerhalb des Architekturmodells zueinander in Beziehung. Dies erfolgt auf Basis von Beziehungs-Metamodellen (BMM), welche die Bausteine jeweils zweier Metamodelle unterschiedlicher Teilmodellsysteme miteinander verknüpfen. Beziehungspatterns (P) unterstützen die Erstellung eines Beziehungs-Metamodells [Sinz95, S. 3].
Ein Beispiel für eine konkrete Schema-Architektur aus dem Kontext betrieblicher Informationssysteme ist das Semantische Objektmodell (SOM) (Abschnitt 4.1, [FeSi13, S. 194 ff.]).

2.2 Grundlagen der Systementwicklung

Der Begriff *Systementwicklung* (synonym: Systems Engineering) bezeichnet allgemein die Spezifikation, Entwicklung, Validierung, das in Einsatz bringen und die Wartung von Systemen [Somm12, S. 316 ff.]. Übertragen auf den Betrachtungsgegenstand *Informationssystem* meint Systementwicklung die Entwicklung betrieblicher Informationssysteme mit Anwendungssystemen als maschinelle Aufgabenträger [FeSi13, S. 9 f.]. Anwendungssysteme beschreiben Lösungsverfahren für die zu automatisierenden IS-Aufgaben und bilden die Aufgaben auf die eingesetzten Rechner- und Kommunikationssysteme ab [FeSi13, S. 479]. Das Lösungsverfahren wird dabei in Form von Anwendungssoftware spezifiziert, die eine oder mehrere Nutzermaschinen unter Zugriff auf eine oder mehrere Basismaschinen realisiert (Abschnitt 2.2.4.2).

Ergebnis des modellbasierten Entwicklungsprozesses in dieser Arbeit ist die spezifizierte Anwendungssoftware (Programmcode) von RESTful SOA. Entsprechend dieser Zielsetzung wird der Begriff der Systementwicklung im Folgenden eng abgegrenzt und umfasst im Sinne des Software Engineerings die Konstruktion von Informationssystemen mit einem softwarebasierten Anwendungssystem als Zielsystem. Den Ausgangspunkt der Entwicklung bildet die IS-Aufgabenebene. Die weiteren Ausführungen fokussieren die Abbildung von Aufgaben in SOM-Geschäftsprozessmodellen auf die Softwarebausteine des Anwendungssystems RESTful SOA. Die Entwicklung von Systemen unterhalb der Abstraktionsebene der Anwendungssoftware, wie z. B. Systemsoftware oder Rechner- und Kommunikationssysteme, wird nicht näher betrachtet.

Die Aufgabe der Systementwicklung wird zunächst anhand eines Modells strukturiert und erläutert (Abschnitt 2.2.1). Die Durchführung der AwS-Entwicklung erfolgt phasenorientiert auf den Beschreibungsebenen der Systementwicklung (Abschnitte 2.2.2 und 2.2.3). Da AwS meist eine hohe Komplexität aufweisen, sind geeignete Hilfsmittel erforderlich, um diese beherrsch-

[1] Strukturmuster, die spezifische Entwurfsentscheidungen dokumentieren und damit Lösungsverfahren für ein bestimmtes Problems beschreiben, werden auch als *Entwurfsmuster* bezeichnet (z. B. [HSW98], [Appl97]). *Architekturmuster* beschreiben den grundlegenden strukturellen Aufbau eines Systems und dessen Gliederung in Teilsysteme (z. B. [Busc00, S. 11 ff.]).

bar zu machen (Abschnitt 2.2.4). Entwicklungsmethodiken stellen hierfür einen methodischen Beschreibungsrahmen sowie geeignete Hilfsmittel bereit (Abschnitt 2.2.5).

2.2.1 Systementwicklung als Aufgabe

Die *Systementwicklung* beschäftigt sich mit der ingenieurmäßigen Entwicklung betrieblicher Anwendungssysteme. FERSTL und SINZ interpretieren die Systementwicklung auf Basis des Strukturmodells der Aufgabe (Abschnitt 2.1.3).

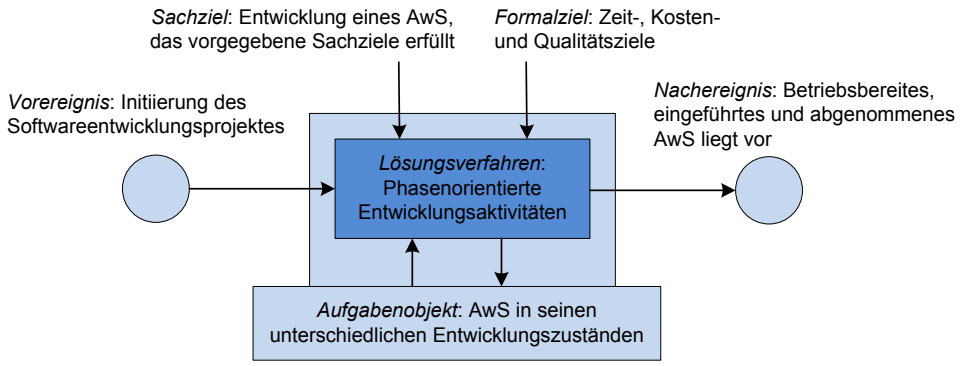


Abbildung 2.7: Aufgabenmodell der Systementwicklung [FeSi13, S. 497]

Abbildung 2.7 visualisiert die Bestandteile der Systementwicklungsaufgabe. Diese werden nun in allgemeiner Form beschrieben (nach [FeSi13, S. 497 f.]):

- *Sachziel* ist die Entwicklung eines AwS, das vorgegebene Anforderungen erfüllt. Die inhaltlichen Anforderungen spezifizieren die Leistung (*was*) des AwS, und qualitative Anforderungen die Vorgaben der Leistungsdurchführung (*wie*).
- *Formalziele* geben bezüglich der Erreichung von Zeit-, Kosten- und Qualitätszielen Anforderungen an die Durchführung der Systementwicklungsaufgabe vor.
- *Aufgabenobjekt* ist „*das zu entwickelnde AwS in seinen unterschiedlichen Entwicklungszuständen*“ [FeSi13, S. 498]. Es umfasst die Software- und Hardwarekomponenten sowie alle weiteren Spezifikationen und Modelle aus der Umgebung des AwS.
- *Vorereignis* ist die Initiierung des Softwareentwicklungsprojektes.
- *Nachereignis* der Softwareentwicklungsaufgabe ist das betriebsbereite, eingeführte und abgenommene AwS.
- Das *Lösungsverfahren* der Systementwicklungsaufgabe spezifiziert die phasenorientierte Durchführung der Entwicklungsaktivitäten (Abschnitt 2.2.3).

Eine Konkretisierung des Aufgabemodells im Hinblick auf die Zielsetzung der vorliegenden Arbeit führt zur Eingrenzung des Aufgabenobjekts bezüglich einer Fokussierung auf die Entwicklung von RESTful SOA. Die inhaltlichen Anforderungen an die Systementwicklung werden durch SOM-Geschäftsprozessmodelle spezifiziert. Die Erfassung qualitativer Anforderungen oder Formalziele ist nicht Gegenstand der weiteren Untersuchungen.

2.2.2 Beschreibungsebenen der Systementwicklung

Die Durchführung der zu automatisierenden IS-Aufgaben erfordert die Entwicklung von Anwendungsprogrammen (Anwendungssoftware) zur Realisierung und Bereitstellung des zugehörigen Lösungsverfahrens. Zwischen den Abstraktionsebenen von IS-Aufgabe und Anwendungsprogramm besteht im Allgemeinen eine große semantische Lücke, die meist nur durch die schrittweise Reduzierung des Abstraktionsgrades überwunden werden kann. Ein klassischer Strukturierungsansatz sieht hierfür vier grundlegende Beschreibungsebenen vor, die bei der Systementwicklung durchlaufen werden [FeSi13, S. 482 f.]:

- Das *Modell des betrieblichen Objektsystems* bildet im Allgemeinen den Ausgangspunkt der Systementwicklung. Eine Spezifikation des Objektsystems kann beispielsweise in Form von SOM-Geschäftsprozessmodellen erfolgen.
- Das *Anwendungsmodell* beschreibt das betriebliche Anwendungssystem als fachliches Lösungsverfahrens für die zu automatisierenden (Teil-)Aufgaben innerhalb des Modells des betrieblichen Objektsystems. Eine Spezifikation des Anwendungsmodells kann z. B. in Form von konzeptuellen Objektschemata und Vorgangsschemata erfolgen.
- Die *Softwarearchitektur* spezifiziert die zu entwickelnde Anwendungssoftware des Anwendungssystems in Form von Komponenten und ihren Beziehungen untereinander (Abschnitt 2.2.4). Ein Beispiel dafür ist die Architektur der RESTful SOA (Abschnitt 5.2).
- Das *Programm* spezifiziert die Implementierung von Komponenten und Beziehungen der Anwendungssoftware in Form von Quellcode.

Die Beschreibungsebenen *Modell des betrieblichen Objektsystems* und *Anwendungsmodell* bilden die *fachlichen Ebenen* der Systementwicklung. Die *softwaretechnischen Ebenen* bestehen aus der *Softwarearchitektur* und dem *Programm*. Der Übergang zwischen fachlicher und softwaretechnischer Ebene erfolgt zwischen Anwendungsmodell und Softwarearchitektur [FeSi13, S. 499]. Die Beschreibungsebenen sind in Abbildung 2.8 des nachfolgenden Abschnitts dargestellt.

2.2.3 Phasen der Systementwicklung

Die phasenorientierte Durchführung von Aktivitäten zur Entwicklung eines AwS spezifiziert das Lösungsverfahren der Systementwicklungsaufgabe [FeSi13, S. 498]. Zielsetzung der Aktivitäten ist dabei eine Überbrückung der semantischen Lücke zwischen den Beschreibungsebenen der Systementwicklung. Entsprechend dieser Interpretation werden korrespondierend zur Aktivitätendurchführung drei Hauptphasen, abgeleitet (Abbildung 2.8).

Die drei zentralen Phasen der Systementwicklung sind Anforderungsanalyse und -definition, Softwaredesign und Realisierung (nach [FeSi13, S. 498 f.]):

- Die zu automatisierenden Aufgaben des „Modells des betrieblichen Objektsystems“ bilden den Ausgangspunkt für die Beschreibung der fachlichen Anforderungen an das Anwendungssystem. Die Aktivität der Spezifikation beider Modelle wird in der Phase *Anforderungsanalyse und -definition (*A*)* durchgeführt.

- Die Spezifikation der „Softwarearchitektur“ des Anwendungssystems („Programmieren im Großen“) erfolgt auf Basis des „Anwendungsmodells“. Die Aktivität des Entwurfs der Struktur der Anwendungssoftware wird in der Phase *Softwaredesign (Softwareentwurf) (*D*)* durchgeführt. Hierbei erfolgt der Übergang von der fachlichen auf die softwaretechnische Ebene.
- Die Spezifikation der Softwarearchitektur bildet die Basis für die eigentliche „Implementierung“ (Programmierung) des Anwendungsprogramms („Programmieren im Kleinen“). Die Aktivität der Programmierung ist Gegenstand der Phase *Realisierung (*R*)*.

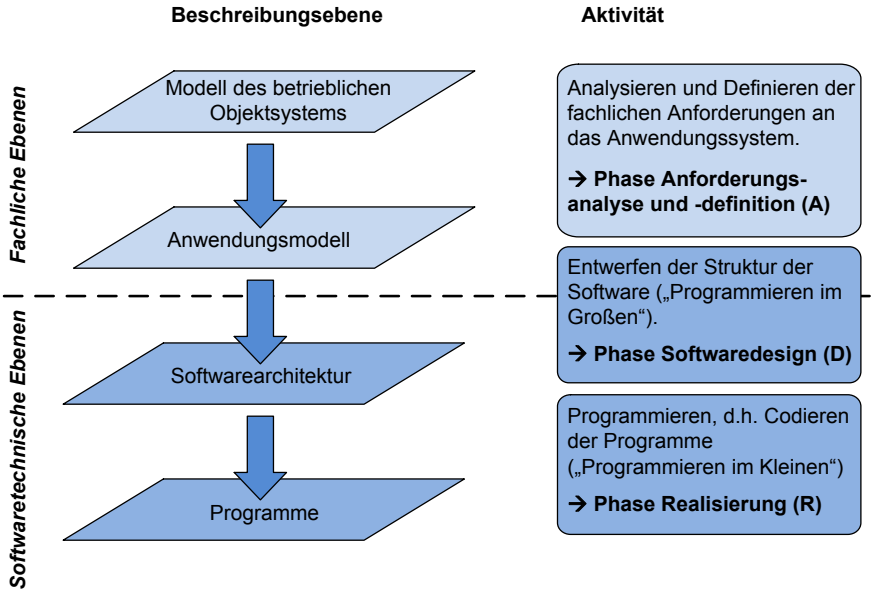


Abbildung 2.8: Aktivitäten der Systementwicklung [FeSi13, S. 499]

Die Phase A, D und R beschreiben die Kernaktivitäten der Systementwicklung. Da eine Entwicklung normalerweise in Form eines Projektes durchgeführt wird, welches zu planen ist, wird die Phase der *Projektplanung* (P) dem Beginn der Systementwicklung vorangestellt. Die *Abnahme und Einführung* (E) des betrieblichen Anwendungssystems beschreibt den Abschluss des Projektes und bildet die letzte Phase [FeSi13, S. 500 f.]. Die Phasen P, A, D, R und E (auch PADRE-Phasen genannt) korrespondieren in ihrer schrittweisen Durchführung mit sequenziellen Vorgehensmodellen wie z. B. dem Wasserfallmodell (z. B. [Kuhr13], [PfAt10, S. 74 f.], [FeSi13, S. 501]).

Die Durchführung der Systementwicklungsphasen wird im Allgemeinen an einem *Vorgehensmodell* ausgerichtet (z. B. [Kuhr13], [Pres10, S. 29 ff.], [Somm12, S. 53 ff.]. Dieses beschreibt den Softwareentwicklungsprozess in einer abstrakten, generischen Form. Ein Vorgehensmodell spezifiziert

- welche Aktivitäten einer Phase,
- in welcher Reihenfolge,
- mit welchem Zielbeitrag pro Phase,
- an welchen Teilen oder am gesamten Aufgabenobjekt

durchzuführen sind [FeSi13, S. 506 f.]. In der Literatur wird eine Vielzahl an Vorgehensmodellen vorgeschlagen, die je unterschiedliche Zielsetzungen verfolgen (z. B. [Somm12, S. 53 ff.], [Pres10, S. 29 ff.], [PfAt10, S. 74 ff.]). Vorgehensmodelle sind Bestandteil von Entwicklungsmethodiken, die ein wichtiges methodisches Hilfsmittel für die ingenieurmäßige Durchführung der Systementwicklungsaufgabe sind (Abschnitt 2.2.5).

WEITERENTWICKLUNG DES BETRIEBLICHEN INFORMATIONSSYSTEMS

Das Ziel der generischen Systementwicklungsaufgabe ist die Entwicklung eines AwS, das vorgegebene Sachziele erfüllt. Als Spezialisierung lassen sich dabei die *Entwicklung* neuer sowie die *Weiterentwicklung* bereits existierender Systeme unterscheiden.

In der Literatur existiert kein einheitliches Verständnis des Begriffs der Weiterentwicklung. Häufig werden hierunter eine Vielzahl an Tätigkeiten, wie beispielsweise die Korrektur von Fehlern in ursprünglichen Systemanforderungen, die Anpassung an geänderte Umweltbedingungen (z. B. technische Umgebung) sowie die Implementierung neuer Anforderungen verstanden ([Somm12, S. 285], [Balz11, S. 530]). Im Software Engineering wird synonym zu den Aufgaben der Weiterentwicklung auch der Begriff *Software-Maintenance* (Softwarewartung) verwendet (z. B. [Somm11, S. 234 ff.], [Pres10, S. 762 ff.], [PfAt10, S. 561 ff.]). BALZERT unterscheidet dabei zwischen den Teilaufgaben der *Wartung* und *Pflege* [Balz11, S. 530]. Wartung bezeichnet die Aktivitäten der Stabilisierung eines Systems durch die Korrektur von Fehlern sowie dessen Optimierung zur Leistungsverbesserung. Als Pflege wird die Anpassung des Systems an geänderte Rahmenbedingungen sowie die Erweiterung dessen Funktionalität aufgrund neuer Anforderungen verstanden [Balz11, S. 533 und 537].

Die Weiterentwicklung betrieblicher Informationssysteme, und folglich auch ihrer Anwendungssysteme wird in dieser Arbeit entsprechend dem phasenorientierten Ablauf der „Neu"-Entwicklung auf den Beschreibungsebenen der Systementwicklung interpretiert. Ausgehend von Änderungen im betrieblichen Objektssystem erfolgt die Entwicklung der angepassten Version eines existierenden Systems *top-down* entlang der fachlichen und softwaretechnischen Ebenen. Dieses Verständnis von Weiterentwicklung geht konform zum eingangs vorgestellten Begriff der *Pflege* von Systemen. Synonym zum Verständnis der „Neu"-Entwicklung und Weiterentwicklung von Systemen werden deshalb im Kontext von betrieblichen Informationssystemen auch die Begriffe *Gestaltung* bzw. *Pflege* verwendet.

Abschließend wird der Begriff der Systemweiterentwicklung in Bezug auf das Untersuchungsobjekt der Arbeit präzisiert. Mit dem Ziel, die Anpassung der Funktionalität von RESTful SOA an veränderte Anforderungen in flexiblen SOM-Geschäftsprozessmodellen zu unterstützen, wird die Weiterentwicklung im Sinne der Pflege von Informationssystemen verwendet. Die Systemweiterentwicklung wird dabei verstanden als die „*Erweiterung oder Anpassung der Spezifikation eines betrieblichen Informationssystems, um dessen Anwendungssystem(e) konsistent auf veränderte Systemanforderungen abzustimmen*".

2.2.4 Strukturierung von Softwaresystemen

2.2.4.1 Softwarearchitektur

Architekturen sind ein grundlegendes Konzept zur Strukturierung von Softwaresystemen. Nach BASS ET AL definiert die Softwarearchitektur eines Systems „*dessen Software-Struktur respektive dessen -Strukturen, dessen Software-Bausteine sowie deren sichtbaren Eigenschaften und Beziehungen zueinander.*“ ([BCK03, S. 3 ff.], Übersetzung durch [Voge09, S. 48]). Je nach Abstraktionsebene spezifiziert eine Softwarearchitektur eine fachliche, (software-)technische oder plattformspezifische Architektur [Voge09, S. 48 und S. 52 f.]. Die Bausteine der Architektur bestimmen einen, anhand festgelegter Kriterien, abgegrenzten Teil des Softwaresystems [Balz11, S. 23 f.]. Je nach Detailgrad beschreiben Bausteine z. B. Teilsysteme, Komponenten oder Klassen des Softwaresystems.

Der Architekturbegriff aus Abschnitt 2.1.6 wird in dieser Arbeit auf die Domäne des Softwaresystems übertragen. Diesem Verständnis folgend legen *Softwarearchitekturen* den strukturellen Aufbau (Bauplan) von Softwaresystemen durch Spezifikation von deren Komponenten und Beziehungen fest, und stellen Hilfsmittel (Konstruktionsregeln) für deren Gestaltung bereit [Sinz13b]. Der Begriff wird somit in einem weiter gefassten Sinne auf Basis eines allgemeineren Verständnisses von Softwaresystemen interpretiert. Die Beschreibung einer Softwarearchitektur erfolgt auf den Abstraktionsebenen des Anwendungsmodells, der Softwarearchitektur (im engeren Sinne) sowie der Programme. Sie korrespondiert folglich mit dem Anwendungssystembegriff der Systementwicklung (Abschnitte 2.1.2 und 2.2). In Anlehnung an die Architektur eines Informationssystems und seinen Beschreibungsebenen wird im Folgenden auch von *Systemarchitekturen* gesprochen.

Häufig bestehen Systemarchitekturen aus einer großen Anzahl von Komponenten und weisen damit eine hohe Komplexität auf. Zur Beherrschung dieser Komplexität bietet sich die Verwendung von Strukturierungskonzepten an. Bei *Schichtenmodellen* entsteht eine stärkere Strukturierung der Komponenten mittels Zuordnung zu verschiedenen Schichten [Balz11, S. 46 f.]. Schichten weisen meist eine Schnittstelle für den äußeren Zugriff auf ihre Komponenten auf. Mit den Schichtenmodellen der *Nutzer- und Basismaschine* sowie dem *ADK-Strukturmodell* werden im Folgenden zwei Strukturierungsansätze vorgestellt.

In der Literatur existiert eine Vielzahl an allgemeinen oder speziellen Definitionen von Softwarearchitektur. Für die nähere Begriffsbestimmung sei auf die weiterführenden Arbeiten verwiesen (z. B. [Balz11, S. 23 ff.], [III11], [Pres10, S. 243 ff.], [Voge09, S. 48 ff.], [PfAt10, S. 58]).

2.2.4.2 Nutzer- und Basismaschine

Das Modell der Nutzer- und Basismaschine beschreibt nach FERSTL und SINZ „*die Struktur einer programmgesteuerten Maschine aus der Außen- und aus der Innensicht*“ [FeSi13, S. 318].

Aus der Außensicht wird die programmgesteuerte Maschine als *Nutzermaschine* bezeichnet. Sie umfasst Operatoren und zugehörige Datenobjekte, die von außen sichtbar und aufrufbar sind. Aus der Innensicht wird die Nutzermaschine durch ein *Programm* realisiert, das seinerseits auf die Datenobjekte und Operatoren der *Basismaschine* zugreift. Dem Nutzer der

programmgesteuerten Maschine ist ausschließlich die *Schnittstelle der Nutzermaschine* zugänglich. Durch diese Kapselung wird von der eigentlichen Realisierung der Nutzermaschine abstrahiert [FeSi13, S. 318 f.]. Das Strukturmodell ist in Abbildung 2.9 dargestellt.

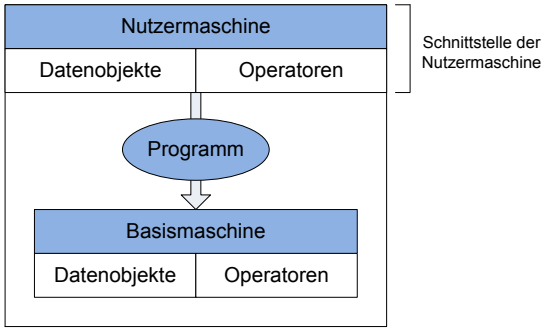


Abbildung 2.9: Nutzer- und Basismaschine [FeSi13, S. 319]

Zur Reduzierung der Komplexität von Systemen ist eine mehrstufige Anordnung der Nutzer-/ Basismaschine möglich. Eine Nutzer-/Basismaschine kann mehrstufig realisiert werden, indem ihre Basismaschine wiederum als Nutzer- und Basismaschine strukturiert wird. Die Basismaschine B_i besteht dann aus einer Nutzermaschine N_{i+1}, Programm P_{i+1} und Basismaschine B_{i+1}. Die Nutzermaschine N_i greift über das Programm P_i somit auf die Schnittstelle der Nutzermaschine N_{i+1} zu (siehe auch [FeSi13, S. 319 f.]).

Ziel des vorgestellten Strukturierungskonzepts ist, neben der Reduzierung von Komplexität einer programmgesteuerten Maschine mittels Schichtenbildung, die Erhöhung der Flexibilität. So können auf einer Basismaschine durch Programme mehrere Nutzermaschinen realisiert werden. Umgekehrt ist auch die Implementierung einer Nutzermaschine für unterschiedliche Basismaschinen möglich [FeSi13, S. 320].

2.2.4.3 ADK-Strukturmodell

Das ADK-Strukturmodell gliedert ein AwS in drei Teilsysteme [FeSi13, S. 321 f.]:

- Anwendungsfunktionen (A)
- Datenverwaltung (D)
- Kommunikation (K)

Der Kommunikationsteil wird wiederum in die Bereiche Kommunikation mit Personen (K_P) und Kommunikation mit Maschinen (K_M) unterteilt [FeSi13, S. 321].

Die Teilsysteme A, D, K_P und K_M lassen sich orthogonal zum Konzept der Nutzer- und Basismaschine gliedern. Somit besteht jedes Teilsystem aus einer Nutzermaschine, einem Programm und einer Basismaschine. Über den Austausch von Nachrichten stehen die Teilsysteme miteinander in Verbindung. Die Kommunikation zwischen Nutzermaschinen wird durch das Programm und die zugrundeliegenden Basismaschinen realisiert [FeSi13, S. 322]. Als Ziele, die mit der ADK-Strukturierung verfolgt werden, sind Flexibilität und Reduzierung der Komplexität von Maschinen sowie Standardisierung und Portierbarkeit zu nennen [FeSi13, S. 322].

2.2.5 Entwicklungsmethodiken

In der Wirtschaftsinformatik stellen Entwicklungsmethodiken das wichtigste methodische Hilfsmittel zur Unterstützung der Durchführung der im Allgemeinen komplexen Systementwicklungsaufgabe dar.
Der Begriff der *Methode* ist im Umfeld der Wirtschaftsinformatik nicht einheitlich definiert (z. B. [Grei04]). Einigkeit herrscht im Verständnis bezüglich der Anwendung eines ingenieurmäßigen Vorgehens bei der Entwicklung von Informationssystemen ([Grei03, S. 956], [FeSi13, S. 133]). Allgemein werden Methoden dabei als eine planbare, begründete Art und Weise des Handels zur Verfolgung einer bestimmten Zielsetzung verstanden ([Grei03, S. 957], [Balz09, S. 53]). Bei der Zielsetzung einer Methode wird zwischen der Erlangung wissenschaftlicher Erkenntnisse oder praktischer Ergebnisse unterschieden ([Dude07], [Grei03, S. 957]). Die Beschreibung eines planbaren, begründeten Vorgehens zur Erreichung der festgelegten Ziele erfolgt in der Disziplin der Wirtschaftsinformatik normalerweise in Form von Vorgehensmodellen (Abschnitt 2.2.3).

In der vorliegenden Arbeit korrespondiert die AwS-Entwicklung mit der Spezifikation von Modellen auf den Beschreibungsebenen der Systementwicklung (Abschnitt 2.2.2). Eine *Entwicklungsmethodik* wird deshalb analog zum Begriff der *Modellierungsmethodik* nach SINZ definiert [Sinz13c, Kapitel 1, S. 9]. Sie umfasst folgende Bestandteile:

- Einen Modellierungsansatz,
- ein Architekturmodell,
- ein Vorgehensmodell und
- ggf. Softwarewerkzeuge.

Das *Architekturmodell* legt den strukturellen Rahmen für den Aufbau des zu entwickelnden Modellsystems fest. Handelt es sich um ein umfassendes Modellsystem, weist es also hohe Komplexität auf, so ist eine Untergliederung in Modellebenen und -sichten sinnvoll (vgl. hierzu das Konzept des generischen Architekturrahmens in Abschnitt 2.1.6).

Die Erstellung des Modellsystems wird durch einen *Modellierungsansatz* methodisch geleitet (Abschnitt 2.1.5). Im Falle einer Strukturierung des Modellsystems in mehrere Modellebenen kann deren Beschreibung jeweils durch die Verwendung eines „Ebenen-spezifischen“ Modellierungsansatzes erfolgen.

Das *Vorgehensmodell* formuliert einen abstrakten, generischen Beschreibungsrahmen des Systementwicklungsprozesses [FeSi13, S. 506]. Demnach strukturiert es den Ablauf der Modellierungstätigkeit innerhalb und, im Falle mehrerer, zwischen den Ebenen der Modellarchitektur.

Der Einsatz von *Softwarewerkzeugen* verfolgt das Ziel der Erhöhung des Automatisierungsgrades der Durchführung der Entwicklungsaufgabe. Je nach Art und Umfang der bereitgestellten Funktionalität unterstützen Werkzeuge dabei die Gesamtaufgabe, oder ein bzw. mehrere Teil-Aufgaben der Systementwicklung (z. B. [Kara13], [Balz09, S. 59 f.]). Für eine Klassifikation von Werkzeugen in der Systementwicklung wird auf die Ausführungen in BALZERT verwiesen [Balz09, S. 59 ff.].

2.3 Modellgetriebene Entwicklung von Softwaresystemen

Die Überwindung der semantischen Lücke zwischen einer fachlichen Problemstellung und der Implementierung eines Softwaresystems als zugehörige Problemlösung ist eine zentrale Herausforderung in der Systementwicklung ([FrRu07, S. 1], [FeSi13, S. 482]). Zur Bewältigung dieser Herausforderung sind insbesondere die Abstraktion und die Strukturierung des betrachteten Gegenstandbereichs in Form von Modellen wichtige Hilfsmittel (z. B. [Balz09, S. 26 ff.], [Kurb90, S. 263], [PeMe06, S. 9 f.]).

Traditionell werden dabei Modelle verwendet, um das zu entwickelnde System auf einer oder mehreren Abstraktionsebenen zu beschreiben. Klassische Anwendungsbereiche sind die Anforderungsanalyse und -definition oder auch die Spezifikation der Struktur von Softwarearchitekturen (z. B. [Rupp09, S. 183 ff.], [Balz09, S. 547 ff.], [Voge09, S. 264 ff.], [Pohl08], 279 ff.]). Diese Art der Modellnutzung innerhalb des gesamten Entwicklungsprozesses, oder einzelnen Teilen davon, wird häufig *modellbasiert* genannt (z. B. [SVE+07, S. 3]). Der Einsatz von Modellen dient hier primär dem Zweck der Dokumentation sowie der Bereitstellung eines Mediums zur Unterstützung der Kommunikation zwischen den beteiligten Subjekten (z. B. [SVE+07, S. 3], [Balz09, S. 80], [Voge09, S. 264 ff.]). Die eigentliche Implementierung des Programms (Codierung) erfolgt auf Basis von Modellen dann meist manuell durch den Softwareentwickler, also in nicht-automatisierter Form.

Ziel der *modellgetriebenen Softwareentwicklung* ist die direkte Erzeugung von Programmcode aus dem spezifizierten Modellsystem. Durch Erhöhung des Abstraktionsgrades soll die Implementierungstätigkeit auf die Ebene der Modelle gehoben werden (z. B. [HaTa06, S. 452], [FrRu07, S. 40 f.]). Begriffe und Konzepte der modellgetriebenen Softwareentwicklung werden in Abschnitt 2.3.1 erläutert. Mit der *Model Driven Architecture* schlägt die Object Management Group ein Rahmenwerk zur Realisierung von modellgetriebenen Entwicklungsansätzen vor (Abschnitt 2.3.2). In der vorliegenden Arbeit erfolgt die Überführung einer fachlichen Problemstellung in eine Systemimplementierung in teilautomatisierter und modellbasierter Form. Das Verständnis der *modellbasierten Systementwicklung* wird in Abschnitt 2.3.3 vorgestellt.

2.3.1 Modellgetriebene Softwareentwicklung

Die Begriffe *modellgetriebene Softwareentwicklung*, *Model Driven Software Development* (MDSD) oder auch *Model Driven Development* (MDD) bezeichnen im Software Engineering die Disziplin der automatisierten Erzeugung von Code aus Modellen (z. B. [HaTa06, S. 452], [Voge09, S. 176], [Balz09, S. 79 f.]). Als weitere Bezeichnung findet sich in der Literatur oftmals auch *Model Driven Engineering* (MDE), welches als Oberbegriff für alle Ansätze dient, in denen Modelle die zentralen Artefakte im Prozess des Software Engineerings darstellen (z. B. [FrRu07], [BCW12, S. 9]). In Bezug auf die verfolgte Zielsetzung der automatisierten Codeerzeugung werden die Begriffe jedoch häufig synonym verwendet [Voge09, S. 176]. Nachfolgend wird deshalb primär von „MDE“ oder „modellgetriebener (Software-)Entwicklung“ gesprochen.

Das *Hauptziel modellgetriebener Entwicklung* ist die Erhöhung des Abstraktionsgrades, auf dem ein lauffähiges Softwaresystem spezifiziert werden kann ([HaTa06, S. 455], [FrRu07, S. 37]). Das Prinzip der Abstraktion wird seit langem im Software Engineering eingesetzt, um

die Komplexität von Softwarebausteinen und den mit ihrer Entwicklung einhergehenden Aufwand zu reduzieren. Als Beispiele seien die Einführung von Assemblersprachen, um von der Ebene des Maschinencodes zu abstrahieren, oder auch die Verwendung von höheren Programmiersprachen (z. B. Fortran oder Cobol) genannt, um Entwickler von der Notwendigkeit einer Spezifikation maschinenspezifischer Befehle „befreien“ zu können ([Schm06, S. 25], [HaTa06, S. 455]). Durch den Einsatz von computerbasierten Technologien wird lauffähige Software aus der definierten höher-abstrakten Systembeschreibung automatisiert abgeleitet. Die Ansätze des MDE führen diese Zielsetzung durch die Einbeziehung weiterer Beschreibungsebenen in die Modellbildung fort und rücken Modelle in den Mittelpunkt des Entwicklungsprozesses (z. B. [PeMe06, S. 9 f.]). In der *MDE Vision* von Softwareentwicklung nach FRANCE und RUMPE

> „[…] *stellen Modelle die primären Entwicklungsartefakte dar, auf die sich Entwickler stützen, um unter Einsatz computerbasierter Technologien Modelle in lauffähige Systeme zu transformieren*“ ([FrRu07, S. 37], eigene Übersetzung).

Darüber hinaus sind Modelle für die Modellersteller und -nutzer der zentrale Betrachtungsgegenstand, um das entwickelte Softwaresystem zu verstehen, mit diesem zu interagieren oder es weiterzuentwickeln [FrRu07, S. 38]. Konform zur MDE Vision ist auch das Verständnis von STAHL ET AL, welche die *modellgetriebene Softwareentwicklung* als „[…] *Oberbegriff für Techniken, die aus formalen Modellen automatisiert Software erzeugen*“ definieren [SVE+07, S. 11]. Die Autoren betonen die automatisierte Softwareerzeugung in einem Transformationsschritt.

Der Begriff *modellgetriebene Softwareentwicklung* lässt sich zusammenfassend anhand folgender Bestandteile charakterisieren ([FrRu07, S. 37], [SVE+07, S. 11], [HaTa06, S. 452], [BCW12, S. 7 f.]):

- Ausgangspunkt der Entwicklung bilden formal beschriebene *Modelle*, die
- durch den Einsatz von *Technologien*
- in ein *lauffähiges Softwaresystem*
- *transformiert* werden.

Die formale Spezifikation der Modelle setzt eine geeignete Notation zur Modellierung der spezifischen Anforderungen in der Problemdomäne voraus. Normalerweise erfolgt eine Modellierung durch den Einsatz von *domänenspezifischen Modellierungssprachen*, kurz DSL (Domain Specific (Modeling) Language). Eine DSL ist eine mit dem Begriffssystem der zu modellierenden Problemdomäne sowie den Eigenschaften des zu implementierenden Softwaresystems abgestimmte Notation (z. B. [Schm06, S. 26 f.], [SVE+07, S. 97 ff.]). Die Anwendung einer DSL setzt die Konstruktion eines zugehörigen Metamodells voraus (vgl. Abschnitt 2.1.5 sowie [SVE+07, S. 30], [Schm06, S. 26 f.]). Bei der modellgetriebenen Entwicklung objektorientierter Systeme, insbesondere im Kontext der Model Driven Architecture, findet häufig auch die (generische) Notation der Unified Modeling Language (UML) Verwendung (z. B. [OMG03], [Kent02], [KWB04], [PeMe06]).

Modelle sollen in der MDE alle relevanten Eigenschaften des Zielsoftwaresystems beschreiben, um dieses automatisiert erzeugen zu können. Damit die Spezifikation von Modellen auf einem

höheren Abstraktionsgrad realisierbar ist, werden die spezifischen Implementierungsdetails der zugrundliegenden Plattform in der eingesetzten *Transformationstechnologie* gekapselt. Weit verbreitet ist der Einsatz von Codegeneratoren zur Durchführung dieser Aufgabe [SVE+07, S. 139 ff.]. Codegeneratoren transformieren die spezifizierten Modelle zur Buildtime in den Quellcode des Softwareprogramms. Diese Art der Übersetzung wird allgemein als Modell-zu-Code-Transformation (Model-to-Code, M2C) bezeichnet. Erfolgt die Erzeugung des Programmcodes nicht direkt, sondern unter Verwendung von Zwischenmodellen, so kommen Modell-zu-Modell-Transformationen (Model-to-Model, M2M) zum Einsatz [SVE+07, S. 33]. Abschließend sei angemerkt, dass im Rahmen von MDE zwar die vollständige Erzeugung eines Programms angestrebt wird, in der Praxis jedoch normalerweise die Ergänzung der abgeleiteten Systemspezifikation durch das manuelle Einfügen von Code notwendig ist (z. B. [PeMe06, S. 135 ff.], [SVE+07, S. 13]).

Die Disziplin der modellgetriebenen Softwareentwicklung steht in der aktuellen Forschung einer Reihe von *Herausforderungen* gegenüber:

- Die Definition problemspezifischer Modellierungsansätze zur zielgerichteten Beschreibung der Quellmodellsysteme auf passender Abstraktionsebene ist eine zentrale Herausforderung (z. B. [HaTa06, S. 458 f.], [FrRu07, S. 44 f.]).
- Zur Unterstützung der *Weiterentwicklung des Softwaresystems* werden Ansätze zum Management und der Evolution von Modellen, Metamodellen und auch eingesetzter Technologien benötigt. Eine wichtige Aufgabe ist hierbei die Transformation, Evolution, Versionierung und Validierung von Modellen (z. B. [HaTa06, S. 457], [SVE+07, S. 391], [Balz09, S. 83]).
- Die *Überbrückung der semantischen Lücke* zwischen der Beschreibung der Problemstellung und der Implementierung des Softwaresystems ist meist hoch komplex und erfordert unterstützende Konzepte zu deren Beherrschung. Auch die Reduzierung hoher Abstraktionsgrade in Modellen stellt die MDE vor große Herausforderungen (z. B. [FrRu07, S. 41 f.]).

Während auf Seiten der Implementierung durch neue Programmiersprachen und Entwicklungsumgebungen Fortschritte bei der Erreichung verfolgter Ziele gemacht wurden, bleiben die oben genannten, zentralen Herausforderungen aus dem Bereich des MDE weiterhin bestehen ([FrRu07, S. 37], [Balz09, S. 83]).

Die Object Management Group schlägt mit der Model Driven Architecture [OMG01] einen Beschreibungsrahmen als MDE-Standardisierungsinitiative vor (Abschnitt 2.3.2). Auf Grund ihrer hohen Bedeutung in Wissenschaft und Praxis und für die Standardisierung von Modellierungskonzepten und Werkzeugen wird die MDA als konkrete Ausprägung von MDE nachfolgend detaillierter beleuchtet. Weitere ähnliche Initiativen sind Microsoft's Software Factories ([GrSh04], [GrSh03]), das Generative Programming [CzEi05], Model Integrated Computing (MIC) [SzKa97] oder auch Generierungstechniken aus dem Bereich des Domain-Specific Modeling (DSM) ([KeTo08], [VöBe13]).

2.3.2 Model Driven Architecture

Die Object Management Group (OMG)[2] spezifiziert und verwaltet offene und plattformunabhängige Standards zur Entwicklung komplexer Softwaresysteme. Im Jahre 2001 wurde von der OMG eine Standardisierungsinitiative zur modellgetriebenen Entwicklung unter dem Begriff der *Model Driven Architecture* (MDA) vorgeschlagen [OMG01]. Die Ziele der MDA sind insbesondere eine höhere Interoperabilität (Herstellerunabhängigkeit durch Standardisierung), Wiederverwendbarkeit und Portierbarkeit (Plattformunabhängigkeit) der spezifizierten Systemkomponenten [OMG03, S. 2]. Zur Erreichung ihrer Ziele definiert die MDA einen Architekturrahmen, der die Trennung plattformunabhängiger und plattformspezifischer Belange durch die Spezifikation eines Systems auf unterschiedlichen Modellebenen vorsieht. Darüber hinaus werden Konzepte zur Transformation von Modellen eingeführt. Als Ergebnis der letzten (finalen) Modelltransformation soll der generierte Quellcode des Softwaresystems vorliegen ([OMG03, S. 3-7], [KWB04, S. 6], [PeMe06, S. 129]).

Der generische Architekturrahmen der MDA unterscheidet die drei grundlegenden, aufeinander aufbauenden **Modellebenen** Computational Independent Model, Platform Independent Model und Platform Specific Model. Diese betrachten das Gesamtsystem aus drei Perspektiven (sog. Viewpoints) (nach [OMG03, S. 2-5 f.], [BCW12, S. 40 f.]):

- Das *Computational Independent Model* (CIM) stellt die Systemanforderungen aus einer fachlichen und anwendungssystemunabhängigen Perspektive dar. Es grenzt die Domäne der Modellbildung anhand deren Diskurswelt und relevanter Umwelt ab und beschreibt damit das Modell des Objektsystems.
- Das *Platform Independent Model* (PIM) spezifiziert die fachliche, plattformunabhängige Aufgabenträgerebene des zu entwickelnden Systems. Das PIM definiert somit Systemverhalten und -struktur, abstrahiert jedoch von einer konkreten technischen Implementierungsplattform.
- Die plattformspezifische Aufgabenträgerebene des Systems wird im *Platform Specific Model* (PSM) spezifiziert. Das PSM beschreibt die Realisierung der Funktionalität des PIM unter Verwendung der Komponenten einer konkreten technischen Plattform.

Die Plattform ist ein zentraler Begriff in der MDA. Unter *Plattform* werden dabei eine Menge von Subsystemen und Technologien verstanden, die ihre Funktionalität durch Schnittstellen bereitstellen [OMG03, S. 2-3]. Der Begriff selbst wird relativ interpretiert und meint zunächst nur ein Modell, welches spezifisch zu den Bestandteilen einer bestimmten Zielplattform formuliert ist ([Kent02, S. 289], [OMG03, S. 6-2 f.]). Die Komponenten und Dienste der Zielplattform beschreibt das *Platform Model (PM)*.

Das PSM auf Ebene N kann wiederum PIM der nachfolgenden Ebene (N+1) sein. Die Abstraktion der Beschreibung eines PIM ermöglicht die Transformation des PIM in mehrere PSM. Die letzte Modellebene der Softwareerstellung beschreibt in der Regel den Quellcode

[2] OMG Specifications. URL: http://www.omg.org/spec/index.htm (Abruf am 25. März 2015).

des Anwendungsprogramms. Von der MDA wird keine Unterscheidung zwischen plattformspezifischem Modell und Code vorgenommen. Es sei an dieser Stelle angemerkt, dass in der Praxis die Codeebene weiterhin eine Sonderrolle einnimmt, da im Allgemeinen eine Anreicherung des abgeleiteten Codes durch manuellen Code notwendig ist, um ein lauffähiges System zu realisieren (z. B. [PeMe06, S. 106], [SVE+07, S. 13]).

Die Überführung von PIM in PSM erfolgt mittels Modelltransformationen. Eine *Transformation* definiert, wie das Modell einer Quellsprache (Quellmodell) in das Modell einer Zielsprache (Zielmodell) automatisiert überführt werden kann [KWB04, S. 24]. In den Prozess können *weitere Informationen*, z. B. Metamodelle, Markierungen, Mappings oder auch ein PM, zur Transformationsdefinition oder Anreichung der Modellspezifikation einfließen [OMG03, S. 2-6 ff.]. Abbildung 2.10 stellt das generische MDA-Pattern der Modeltransformation nach [OMG03, S. 2-7] in einstufiger (links) und mehrstufiger Form (rechts) dar.

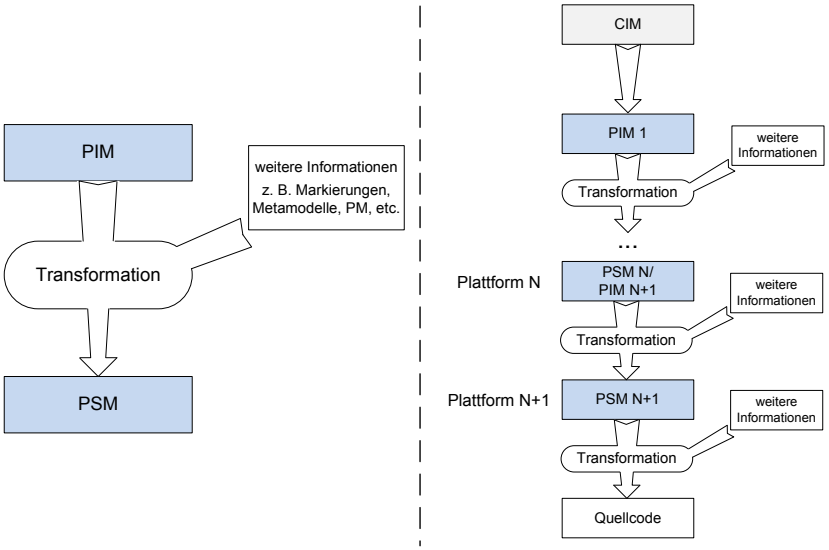


Abbildung 2.10: Einstufiges und mehrstufiges MDA Pattern (in Anlehnung an [OMG03, S. 6-1 ff.])

Entsprechend dem Konzept der Nutzer-/Basismaschine beschreibt das PIM die Nutzermaschine einer Ebene anhand festgelegter Operatoren und Datenobjekte (Abschnitt 2.2.4.2). Mit dem PSM wird die Nutzung der Basismaschine einer Ebene aus dem PIM transformiert. Die MDA spricht in diesem Zusammenhang von der virtuellen Maschine des PIM [OMG03, S. 2-6].

Für die *Transformation* zweier Modelle (PIM und PSM) schlägt die MDA eine Reihe von sogenannten **MDA Mappings** vor [OMG03, S. 3-2 f.]. Auf Schemaebene und Metaebene wird zwischen folgenden grundlegenden Mappings unterschieden:

- *Model Type Mapping*: Zwischen den Metaobjekten der Quell-Modellierungssprache (PIM) wird ein Mapping (Abbildungsbeziehung) auf entsprechende Metaobjekte der Ziel-Modellierungssprache (PSM) definiert (Definition auf Meta-Ebene). Eine Realisierung des Model Type Mapping erfolgt meist durch ein sogenanntes *Metamodel Mapping*. Dieses Vorgehen setzt ein gemeinsames Meta-Metamodell auf PIM- und PSM-Ebene voraus [OMG03, S. 3-2 f.].

- *Model Instance Mapping*: Die Transformation basiert auf der Definition von Mappings, die spezifizieren, wie Elemente des Quellmodells in zugehörige Elemente des Zielmodells überführt werden (Definition auf Schemaebene) [OMG03, S. 3-3 f.].

Eine Kombination der Transformationsarten von Elementen der Schema- und Metaebene ist möglich [OMG03, S. 3-4]. Als Erweiterung dieser beiden grundlegenden Transformationsarten werden drei weitere Ansätze vorgeschlagen (nach [OMG03, S. 3 ff.]):

- *Model Marking*: Durch Markierung von Modellelementen mit Metainformationen wird die Transformation gesteuert. Markierte Elemente können z. B. Metaobjekte, Objekttypen und Beziehungen oder Eigenschaften sein. Die Auswahl und Abbildung von Elementen des Quell-, auf Elemente des Zielmodells erfolgt auf Basis von *Transformationsregeln* anhand der vergebenen Marken.
- *Pattern Application*: Bei der Transformation des Quellmodells können Entwurfsmuster zur Spezifikation der Struktur des Zielmodells eingesetzt werden. Diese werden zusammen mit der typbasierten Modelltransformation oder dem Marking eingesetzt.
- *Template*: Templates stellen einen Rahmen bereit, der durch Parametrisierung die Spezifikation konkreter Elemente des Zielmodells ermöglicht. Der Einsatz von Templates steht häufig in Zusammenhang mit der Codegenerierung, also der Durchführung einer Modell-zu-Code-Transformation.

Die verschiedenen Transformationsarten sind untereinander kombinierbar. Darüber hinaus ist die Einbeziehung weiterer zusätzlicher Informationen möglich. Im *Model Merging* werden mehrere PIMs, die verschiedene Aspekte des Systems beschreiben, in einem Transformationsschritt in das PSM zusammengeführt [OMG03, S. 3-12 f.].
Anmerkung: Die vorgestellten Konzepte fokussieren die Transformation von PIM und PSM und nehmen nicht auf die Ebene des CIM Bezug. Als mögliche Erklärung hierfür sei auf den Abstraktionsgrad hingewiesen, der für eine automatisierte Durchführung der Modelltransformation vorausgesetzt wird. Das CIM modelliert die fachlichen Anforderungen der Problemstellung, die zur Ebene des PIM meist eine große Abstraktionslücke aufweist. Demzufolge ist ein Erzeugen des Softwaresystems aus dem CIM im Allgemeinen nicht automatisiert möglich und damit nicht durchgängig in Form von Transformationen beschreibbar.

Zur Spezifikation des Architekturrahmens schlägt die MDA die Verwendung von OMG Standards als technologische Basis vor (z. B. [Kent02, S. 289], ([OMG03], S. 7-1f), [BCW12, S. 45]). Aus diesem Grund empfiehlt die MDA die *Meta-Object Facility* (MOF) als Meta-Metamodell [OMG13a] sowie die *Unified Modeling Language* (UML) als Notationssprache für die Modellierung [OMG11c]. Daneben wird von der OMG der Einsatz weiterer Spezifikationen wie z. B. der *Object Constraint Language* (OCL) [OMG14] oder *XML Metadata Interchange* (XMI) [OMG13b] empfohlen. Unabhängig von diesen Vorschlägen ist die MDA jedoch neutral bezüglich des Einsatzes konkreter Modellierungsansätze oder Technologien konzipiert.

2.3.3 Modellbasierte Systementwicklung

Die Entwicklung betrieblicher Anwendungssysteme wird in der vorliegenden Arbeit auf allen Beschreibungsebenen der Systementwicklung betrachtet (Abschnitt 2.2.2). Ausgehend von den vorausgegangenen Ausführungen des Abschnitts 2.3 wird der Begriff der *modellbasierten Systementwicklung* definiert. Die Definition nimmt Bezug auf die Herausforderung der Überwindung der semantischen Lücke zwischen fachlichen Anforderungen der Problemstellung und der Systemimplementierung als Problemlösung.

Ausgangspunkt der Entwicklung ist das betriebliche Objektsystem, welches in Form von SOM-Geschäftsprozessmodellen beschrieben wird. Als Ergebnis eines modellgetriebenen Vorgehens soll die Spezifikation des betrieblichen Anwendungssystems, das als RESTful SOA realisiert ist, erzeugt werden. Voraussetzung für eine durchgängige und automatisierte Erzeugung des Anwendungssystems wäre, dass Quellmodelle und Transformationsdefinitionen alle notwendigen plattformspezifischen Details kapseln, um den Detailgrad der Implementierungsebene beschreiben zu können. Ein Erreichen dieser Vorgabe ist im Allgemeinen jedoch nur eingeschränkt möglich, da auf Aufgabenebene und (fachlicher) Aufgabenträgerebene durch den Entwickler Entscheidungen bezüglich der Automatisierbarkeit von Aufgaben und der strukturellen Überarbeitung fachlicher Modelle zu treffen sind (siehe Abschnitte 2.3.1 und 2.3.2).

Eine Einbeziehung nicht-automatisierbarer Entwurfsschritte in die modellgetriebene Entwicklung betrieblicher Anwendungssysteme wird im Kontext der vorliegenden Arbeit als *modellbasierte Systementwicklung* definiert:

> *„In der modellbasierten Systementwicklung wird das Modell des betrieblichen Objektsystems als Abfolge von automatisierten Transformations- und nicht-automatisierten Überarbeitungs- und Anreicherungsschritten in die Spezifikation eines betrieblichen Anwendungssystems überführt.“*

Der Begriff *Systementwicklung* nimmt Bezug auf das Konzept der Systementwicklungsaufgabe, welche das umfassende Ziel der Entwicklung betrieblicher Informationssysteme und ihrer AwS verfolgt. Die Aufgabendurchführung schließt folglich die Überwindung aller Beschreibungsebenen durch ein phasenorientiertes Lösungsverfahren mit ein. Der Begriff *modellbasiert* nimmt Bezug auf den Automatisierungsgrad der Systementwicklungsaufgabe, und betont ihre *teilautomatisierte Durchführung*. Modelle bilden nach diesem Verständnis weiterhin das zentrale Artefakt der Entwicklung und sind Input für die Modelltransformation zwischen den Beschreibungsebenen im Sinne eines modellgetriebenen Vorgehens. Das Lösungsverfahren zur Überwindung der semantischen Lücke sieht eine Folge von *Transformations-* sowie *Überarbeitungs-* und *Anreicherungsschritten* als grundsätzliche Vorgehensweise vor. Innerhalb einer Modellebene wird die abgeleitete Spezifikation vertieft und um weitere Details angereichert. Dies geschieht beispielsweise in Form von Entwurfsentscheidungen und erfordert die Einbeziehung des Modellierers. Die angereicherten Modelle sind wiederum Input für den nächsten Transformationsschritt. Das Konzept der modellbasierten Systementwicklung visualisiert Abbildung 2.11.

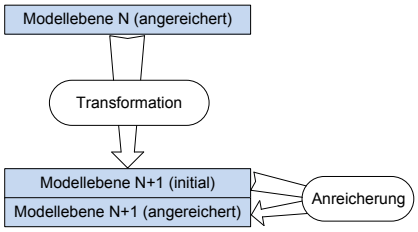


Abbildung 2.11: Konzept der modellbasierten Systementwicklung (in Anlehnung an [OMG03, S. 6-1])

2.4 Zusammenfassung

Betriebliche Informationssysteme bilden den Betrachtungsgegenstand der vorliegenden Forschungsarbeit. Das wichtigste methodische Hilfsmittel zur Beherrschung der Komplexität in IS sind Modelle, die das zentrale Artefakt für ihre Analyse, Gestaltung und Pflege darstellen. Der Modellierer wird bei der Modellsystemerstellung durch die Konzepte Metapher und Metamodell methodisch geleitet. Zur Strukturierung komplexer Modellsysteme untergliedert der generische Architekturrahmen diese in Modellebenen und Modellsichten.

Die Entwicklung betrieblicher Anwendungssysteme erfordert ein ingenieurmäßiges Vorgehen zur Überbrückung der semantischen Lücke zwischen Aufgaben- und Aufgabenträgerebene. Durch die Bildung von Beschreibungsebenen wird der Entwickler bei der Durchführung der Systementwicklungsaufgabe unterstützt. Das phasenorientierte Lösungsverfahren sieht eine schrittweise Reduzierung des Abstraktionsgrades vom Modell des Objektsystems (Problemstellung) hin zum entwickelten Anwendungsprogramm (konstruierte Problemlösung) vor. Der Einsatz von Strukturierungskonzepten unterstützt die Komplexitätsbewältigung des zu erstellenden Softwaresystems. Für die systematische Durchführung der Systementwicklungsaufgabe geben Entwicklungsmethodiken einen strukturellen Rahmen vor.

Die modellgetriebene (Software-)Entwicklung verfolgt das Ziel der Erzeugung von lauffähiger Software aus Modellen. Der Grad der automatisierten Erzeugung des Zielsystems ist dabei ein wichtiges Abgrenzungsmerkmal für Ansätze des MDE. Als Initiative zur Standardisierung modellgetriebener Ansätze formuliert die MDA einen Beschreibungsrahmen für die Gestaltung von Modellarchitekturen und die Modelltransformation. Die semantische Lücke zwischen den IS-Beschreibungsebenen stellt die aktuelle Forschung im Hinblick auf die Zielerreichung einer automatisierten Erzeugung von Programmen aus (fachlichen) Modellen jedoch weiterhin vor große Herausforderungen.

Die Bestimmung des Begriffs der modellbasierten Systementwicklung als Entwicklungsart und seine Unterscheidung von weiteren Prinzipien der modellgetriebenen Softwareentwicklung ordnen die Arbeit thematisch ein. Die methodisch geleitete Überwindung der Abstraktionslücke zwischen dem betrieblichen Objektsystem und dem daraus entwickelten Programm erfolgt dabei teilautomatisiert. Aufgrund der unterschiedlichen Abstraktionsgrade zwischen den IS-Modellebenen ist eine durchgängig automatisierte Entwicklung des Zielsystems nicht realisierbar. Ein solches Vorgehen würde die formale Beschreibung von Entwurfsentscheidungen voraussetzen. Hieraus resultiert die Forderung an die zu konstruierende Entwicklungsmethodik dieser Arbeit, dass diese den Systementwickler bei der Durchführung nicht-automatisierter Entwicklungsaufgaben anleitet und dazu einen passenden methodischen Rahmen vorgibt.

3 Service-orientierte Architekturen

Ziel des dritten Kapitels ist die Einführung und Erläuterung der Grundlagen von RESTful SOA, die in der vorliegenden Arbeit die Zielarchitektur der zu entwickelnden Anwendungssysteme darstellt. Zunächst werden der Begriff der service-orientierten Architektur bestimmt und ihre Merkmale, Bestandteile und ihr grundlegender Aufbau erarbeitet. Mit dem REpresentational State Transfer (REST) wird anschließend ein Architekturstil zur Realisierung von verteilten Systemen erläutert und es werden verbreitete Technologien und Standards zu dessen Implementierung in einem Exkurs vorgestellt. Abschließend wird die RESTful SOA als Realisierungsform verteilter und flexibler betrieblicher Anwendungssysteme konzipiert und beschrieben.

3.1 Grundlagen service-orientierter Architekturen

Service-orientierte Architekturen (SOA) sind das Ergebnis der service-orientierten Gestaltung verteilter betrieblicher Anwendungssysteme. Seit der ersten Veröffentlichung des Begriffs im Jahre 1996 ([ScNa96], [Schu96]) werden die technischen und wirtschaftlichen Ziele und Auswirkungen, die mit der Anwendung des SOA-Konzepts einhergehen, sowohl in der Wissenschaft als auch in der Praxis intensiv diskutiert (z. B. [KaKr08], [VLA09]). Dabei erfolgt eine Abgrenzung des Begriffs *service-orientierte Architektur* in der Literatur sehr unterschiedlich [ErSi08]. Die Bandbreite der vorgeschlagenen Definitionen reicht von einer Fokussierung auf die Entwicklung konkreter Architekturen und Technologien (z. B. [Arsa04], [KBS05]), über die Steuerung und Kontrolle von SOA-Projekten (Governance) (z. B. [KSH08], [Kell07, S. 289 ff.]), bis hin zur Strukturierung der Unternehmensorganisation (z. B. [BBW+05]).

Bei der Betrachtung von SOA wird im weiteren Verlauf der Arbeit die Perspektive des Systementwicklers auf die Gestaltung von AwS eingenommen. Die Durchführung der Systementwicklungsaufgabe bildet somit den Bezugsrahmen einer Untersuchung der Begrifflichkeiten, Merkmale und Konzepte von SOA.

3.1.1 Der Begriff der service-orientierten Architektur

Im Allgemeinen wird unter einer service-orientierten Architektur ein *Architekturkonzept* (synonym: Architekturtyp) verstanden, das aus der Ausrichtung der Systementwicklungsaufgabe auf das service-orientierte Paradigma[3] (Service-Orientierung) bei der Gestaltung verteilter betrieblicher Anwendungssysteme entsteht ([ABB+09, Präambel], [Erl08b, S. 38]). Die Teilsysteme dieser verteilten, service-orientierten AwS sind lose gekoppelt und bieten eine spezifische Dienstleistung (Service) an (z. B. [ECP+13, S. 34], [ABW11, S. 75], [Josu08, S. 15]).

Das Ergebnis der Anwendung von Service-Orientierung auf die Architektur von AwS ist somit das *Konzept* der service-orientierten Architektur, die sich aus einer Menge von *Services*

[3] Denkmuster, Schema [Dude07, S. 996]

(synonym: *Dienste*) zusammensetzt. Jeder Service kapselt eine definierte fachliche Funktionalität und stellt diese in Form von Operationen über eine standardisierte, implementierungsunabhängige Schnittstelle zur Verfügung (z. B. [Balz11, S. 201], [Voge09, S. 152]). Die Nutzung eines Services erfolgt i. d. R. durch den Austausch von asynchronen Nachrichten ([Josu08, S. 45], [Erl08b, S. 43 ff.]). Die fachliche Funktionalität des Services sowie die technischen Details seiner Nutzung werden in einem Service-Vertrag spezifiziert ([Josu08, S. 35 ff.], [Erl08b, S. 127 ff.]). Das Konzept der SOA und veröffentlichter Services ist folglich unabhängig vom Einsatz bestimmter Technologien und Standards (z. B. [Melz10, S. 9], [ABB+09], [Papa08, S. 22 f.], [Sied07, S. 110]).

Eine verbreitete Realisierungsform von SOA stellen Web-Service-Architekturen dar, deren Dienste basierend auf dem Einsatz von Web-Service-Technologien entwickelt werden (z. B. [Melz10, S. 63 ff.], [BHM04], [Góme13], [SeBi13]). Den Kern der Spezifikation von *Web-Services* bilden die Standards SOAP [MiLa07], als Nachrichtenformat der Kommunikation zwischen Diensten, und WSDL (Web Service Description Language) [CMR+07], als Sprache zur Beschreibung von Dienstschnittstellen [SeBi13]. Mit dem Architekturstil REST wird in der Literatur zudem eine weitere Alternative zur Realisierung von SOA intensiv diskutiert (z. B. [ECP+13], [WoBe13], [Alge07, S. 777 ff.]). Eine genauere Untersuchung von *RESTful SOA* erfolgt in den Abschnitten 3.2 und 3.3.

Den beteiligten „Komponenten“ oder auch Teilnehmern in einer SOA werden i. d. R. **Rollen** zugeordnet ([FeSi13, S. 438 f.], [Melz10, S. 14 ff.]). Im Zentrum stehen dabei die Rollen *Service-Provider* (Anbieter) und *Service-Requester* (Nutzer, Konsument). Auf Basis der veröffentlichten Schnittstellenspezifikation des Services (Service-Description) kommuniziert der Service-Requester mit dem Service-Provider, um eine bestimmte Dienstleistung zu konsumieren. Als dritte Rolle wird ferner der *Service-Broker* (Vermittler) in der Literatur genannt. Dieser unterstützt das Finden von speziellen Dienstleistungen in einem *Service-Registry* (Dienstverzeichnis) (z. B. [OASI12, S. 28 f.], [Melz10, S. 14 ff.], [EAA+04, S. 26 ff.], [BHM04, S. 68 ff.]). In REST erfolgt das Entdecken (Discovery) oder Bereitstellen neuer Dienste dagegen normalerweise über das Konzept der Hypermedia zusammen mit selbstbeschreibenden Service-Schnittstellen (vgl. Abschnitt 3.3.2, [PaPa13], [Tilk11, S. 213 f.], [Alla10, S. 251 ff.], [WPR10, S. 243 ff.], [PZL08, S. 812]). Der Service-Broker wird im weiteren Verlauf nicht weiter thematisiert.

ZIELE VON SOA

Mit der Gestaltung von betrieblichen Anwendungssystemen nach dem Konzept der SOA sind im Allgemeinen hohe Erwartungen an die Flexibilität und die Interoperabilität ihrer verteilten Teilsysteme verknüpft ([ERM10, S. 18], [Erl08b, S. 55 ff.]). So zielt die Standardisierung der Schnittstellen von lose gekoppelten Services auf die Realisierung von Potenzialen bezüglich der flexiblen Nutzung von Services in verschiedenartigen betrieblichen Anwendungsfällen und folglich auf die Erhöhung der Wiederverwendung ([Star11, S. 314], [Grah08, S. 23 f.]). Damit einhergehend wird die Realisierung von Potenzialen in Bezug auf die *Anpassbarkeit* (evolutionäre Weiterentwicklung) von Teilsystemen oder des gesamten AwS gesehen (z. B. [Melz10, S. 38 ff.], [Josu09, Wertesystem]).

Aus der betriebswirtschaftlichen Perspektive wird eine verbesserte Abstimmung zwischen Zielen, Geschäftsprozessen und Anwendungssystemen innerhalb der Unternehmensarchitektur angestrebt. Zusammen mit der Realisierung einer erhöhten AwS-Flexibilität wird insgesamt die Steigerung der Wettbewerbsfähigkeit sowie das Erzielen geschäftlichen Mehrwerts für die Unternehmung vom SOA-Einsatz erwartet (z. B. [Star11, S. 313 f.], [Erl08b, S. 61 ff.], [StTi07b, S. 10 f.], [KBS05, S. 239 ff.]).

3.1.2 Merkmale service-orientierter Architekturen

Die Entwicklung von service-orientierten Architekturen befasst sich mit der Gestaltung von Systemlandschaften zur Realisierung verteilter betrieblicher Anwendungssysteme [Sied07, S. 110]. Ein verteiltes betriebliches Anwendungssystem, bzw. die korrespondierende Systemlandschaft, besteht aus einer Menge autonomer (Teil-)Systeme, deren bereitgestellte Dienstleistungen die kooperative Durchführung von IS-Aufgaben unterstützen [FeSi13, S. 483 f.].

Den Ausgangspunkt für die Bestimmung der Merkmale von SOA bildet die Definition nach SIEDERSLEBEN. Nach dem Verständnis des Autors lassen sich SOA im Wesentlichen durch die drei Merkmale *Komponentenorientierung*, *lose Kopplung* und *Workflow* charakterisieren [Sied07, S. 111]. Demnach bestehen service-orientierte Architekturen aus einer Menge von Komponenten (Teilsystemen), die wiederum eine Menge von Services mit klar definierten Schnittstellen veröffentlichen (Komponentenorientierung). Die Services selbst sind untereinander lose gekoppelt und kommunizieren in asynchroner Form über den Austausch von Nachrichten. Die Steuerung des Ausführungsablaufs von Services wird innerhalb der SOA von einer eigenen Komponente verwaltet (Workflow) [Sied07, S. 112 f.].

Neben diesen gestaltungsorientierten Merkmalen wird in der Literatur auch die *fachliche Ausrichtung auf Geschäftsprozesse* als viertes SOA-Kennzeichen definiert (z. B. [BWB11, S. 187], [Star11, S. 314 ff.], [Melz10, S. 33], [Grah08, S. 8 f.], [PaHe07, S. 390], [TLD+07, S. 5]). Die Entwicklung von SOA und ihrer lose gekoppelten Services dient dem Zweck der Unterstützung von Aufgaben in Geschäftsprozessen betrieblicher Systeme. Aus Sicht der Systementwicklung bilden Geschäftsprozesse also den Ausgangspunkt für die Top-Down-Spezifikation der (fachlich) funktionalen und auch nicht-funktionalen Anforderungen an die Services der SOA.

Die vorangegangenen Erläuterungen zum Begriff service-orientierte Architektur werden in folgender Charakterisierung zusammengefasst, die den weiteren Ausführungen zugrunde liegt:

„*Eine service-orientierte Architektur ist ein Architekturkonzept zur Gestaltung betrieblicher Anwendungssysteme als verteilte Systeme. Zur Unterstützung von Geschäftsprozessen veröffentlicht das service-orientierte AwS eine Menge von Services, welche eine fachliche Funktionalität über ihre standardisierten Schnittstellen bereitstellen. Die Durchführung der Systemgestaltungsaufgabe folgt den Prinzipien der Komponentenorientierung und losen Kopplung in Verbindung mit einer ausgelagerten Ablaufsteuerung*.“

3.1.3 Beschreibungsebenen service-orientierter Architekturen

Zur tiefergehenden Analyse und Gestaltung von SOA bietet sich ihre Untergliederung in Beschreibungsebenen an (Abschnitt 2.2.2). Als grundlegende Perspektiven kann hier zwischen

Prozess-, Service- und Technikmodellebene unterschieden werden [Sied07, S. 111]. Diese Modellebenen werden in Abstimmung mit dem Begriffsverständnis der vorliegenden Arbeit als Geschäftsprozess-, Service- und Implementierungsmodellebene bezeichnet (Abbildung 3.1).

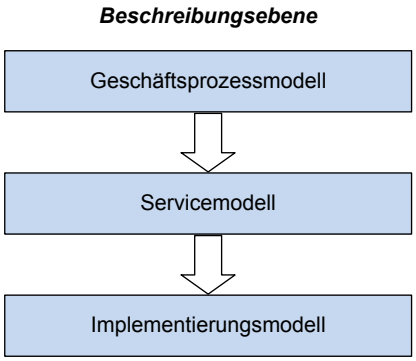


Abbildung 3.1: SOA-Modellebenen (in Anlehnung an [Sied07, S. 111])

- *Geschäftsprozessmodell*: Das betriebliche Objektsystem wird in Form eines Modells der unternehmerischen Geschäftsprozesse erfasst. Es bildet den Ausgangspunkt der Systementwicklung und beschreibt die fachlichen Anforderungen der zu entwickelnden SOA in Form von Aufgaben.
- *Servicemodell*: Die Automatisierung oder Teilautomatisierung der Aufgaben eines Geschäftsprozesses erfolgt durch die Services der SOA. Hierzu werden ihr grundlegender Aufbau und ihre Spezifikation in einem Servicemodell erfasst.
- *Implementierungsmodell*: Die plattformspezifische Realisierung der SOA und ihrer Komponenten beschreibt das Implementierungsmodell. Auf dieser Ebene erfolgt die Wahl von Technologien für die Entwicklung der konkreten Architektur.

Die vorgestellten SOA-Modellebenen strukturieren service-orientierte Architekturen nach den Beschreibungsebenen der Systementwicklung. Das Ziel des Servicemodells ist dabei die Beschreibung der Zuordnung von Aufgaben zu Services in einem plattformunabhängigen Architekturmodell. Nach diesem Verständnis korrespondiert die Ebene des Servicemodells mit dem Anwendungsmodell und dem Modell der Softwarearchitektur (Abschnitte 2.2.2 und 2.3.2).

3.1.4 Klassifikation von Services

Services stellen innerhalb einer SOA verschiedenartige Funktionalität zur Unterstützung der Geschäftsprozessdurchführung in betrieblichen Systemen bereit. Als Basis für eine Strukturierung des Aufbaus von SOA wird im Folgenden eine Klassifizierung von Services anhand ausgewählter Eigenschaften vorgenommen. Auch wenn in der Literatur keine allgemeingültige Einordnung von Services existiert, lassen sich dennoch drei **Grundtypen von Services** identifizieren (z. B. [Josu08, S. 81 ff.], [Erl08b, S. 43 ff.], [Papa08, S. 12 ff.], [KBS05, S. 69 ff.]).

- *Basis-Services* (oder *Entity-Services*) kapseln die Funktionalität für die Verwaltung persistenter Daten oder die Bereitstellung von Geschäftslogik (z. B. [Josu08, S. 82 ff.], [Erl08b, S. 44], [KBS05, S. 70 ff.]).

- *Composed* Services (oder *Task-Services)* implementieren das Lösungsverfahren eines geschäftlichen Vorgangs und greifen hierzu steuernd auf eine Menge von Basis- oder Composed Services zu (z. B. [Josu08, S. 87 ff.], [Erl08b, S. 44 f.]). Die Bildung eines höheren Dienstes durch die koordinierten Aufrufe mehrerer Dienste wird auch als *Komposition* bezeichnet [Erl08b, S. 39 f.].
- *Prozess-Services* repräsentieren den Workflow zur Durchführung eines Geschäftsprozesses (z. B. [Josu08, S. 91 ff.], [Erl08b, S. 45], [KBS05, S. 79 ff.]). Zur Realisierung des Ablaufs eines Geschäftsprozesses komponieren sie eine Menge von Services. Prozess-Services stehen in Interaktion mit den Nutzern der SOA.

Die Klassifizierung von Services erfolgt somit aus verschiedenen Perspektiven, die den Fokus auf technische oder fachliche Kriterien richten.

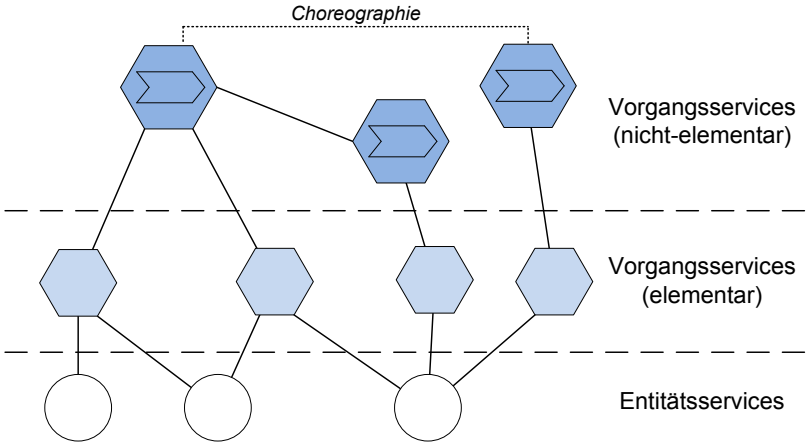


Abbildung 3.2: Servicemodell (in Anlehnung an [KrSi11, S. 294])

Im Folgenden wird die **Klassifizierung von Diensten** nach dem **Servicemodell** von KRÜCKE und SINZ eingeführt [KrSi11, S. 294]. Nach diesem Modell werden Services auf Basis des Modells der betrieblichen Aufgaben (Abschnitt 2.1.3) in *Entitätsservices* und *Vorgangsservices* untergliedert (Abbildung 3.2). Die Nutzung des Aufgabenmodells als methodische Basis erlaubt eine Abgrenzung und Klassifizierung von Services unabhängig von ihrer Realisierung in einer konkreten Architektur.

- *Entitätsservice:* Der *Entitätsservice* stellt eine Schnittstelle zur Verwaltung (der Teilmenge) eines Aufgabenobjekts zur Verfügung. Über die Operationen der Service-Schnittstelle sind die Attribute und Funktionen des Aufgabenobjekts von Service-Konsumenten zugreifbar. Damit kapseln die Entitätsdienste eines Aufgabenobjekts die Realisierung der Datenverwaltung sowie die zugehörige betrieblichen Geschäftslogik [KrSi11, S. 295].
- *Vorgangsservice*: Das Zusammenwirken einer Menge von Entitätsdiensten zur Realisierung der Lösungsverfahren von Aufgaben steuern *elementare Vorgangsservices*. Diese hierarchische Koordination von Services zur Bereitstellung „höherer“ Funktionalität in Form eines weiteren Services wird als *Orchestrierung* bezeichnet (z. B. [Hohp07, S. 444]). Zur Steuerung des Ausführungsablaufs in einem Vorgangsnetz wird eine Menge von Vorgangsservices von *nicht-elementaren Vorgangsservices* orchestriert [KrSi11, S. 295 f.].

Die Durchführung eines Geschäftsprozesses erfordert das Zusammenwirken der definierten Vorgangsnetze bzw. der nicht-elementaren Vorgangsservices der unterstützenden SOA. Da nicht-elementare Vorgangsservices autonom sind, erfolgt die Abstimmung zwischen diesen in nicht-hierarchischer Form. Die Koordination zwischen nicht-elementaren Services wird als *Choreographie* bezeichnet (z. B. [SBB+11, S. 296], [Josu08, S. 121 f.], [Hohp07, S. 443 f.]).

Das Aufgabenmodell ist in der SOM-Methodik die methodische Grundlage zur Beschreibung von Geschäftsprozessen (Abschnitt 4.1.2). In der vorliegenden Arbeit erfolgt deshalb eine Klassifizierung nach dem Servicemodell von KRÜCKE und SINZ.

3.2 REpresentational State Transfer

Der Architekturstil *REpresentational State Transfer* (REST) wurde im Jahre 2000 von FIELDING im Rahmen seiner Dissertation vorgeschlagen [Fiel00]. FIELDING untersucht in seiner Arbeit Architekturentwürfe von netzbasierten Anwendungen mit dem Ziel, Bedingungen zur Klassifikation von Architekturen verteilter *Hypermedia-Systeme*[4] zu definieren. Auf Basis von architektonischen Richtlinien werden Architekturstile bestimmt, welche die Auswahl und Entwicklung von konkreten Architekturen für gegebene Anwendungsszenarien unterstützen ([Fiel00, S. 2], [Mint05, S. 63]). Seit seiner Einführung wird REST in Wissenschaft und Praxis zunehmend als Alternative zur Gestaltung der Architekturen von verteilten webbasierten Anwendungssystemen diskutiert und eingesetzt (z. B. [Tilk11], [Schr11], [PZL08], [Over07], [RiRu07]).

Im Folgenden wird unter einem *Architekturstil* ein Mittel zur abstrakten und technologieunabhängigen Beschreibung von Architekturen anhand einer gegebenen Menge architektonischer Bedingungen verstanden. Das Ziel eines Stils ist dabei, durch seine aufgestellten Bedingungen „*die Rollen oder funktionellen Eigenschaften von Elementen der Architektur und den Beziehungen zwischen diesen Elementen zu beschränken […]*“ ([Fiel00, S. 13], eigene Übersetzung). Ein Stil dokumentiert folglich eine bewährte Vorgehensweise zur Strukturierung von Architekturen [Voge09, S. 34].
Als konkreten Anwendungsfall schlägt FIELDING den Architekturstil REST für die Gestaltung der Architektur des World Wide Web (WWW) vor [Fiel00, S. 76 ff.]. Diese stellt aktuell die wohl bekannteste Implementierung eines verteilten Hypermedia-Systems dar.

REST verfolgt als **Zielsetzung** primär eine Minimierung der Reaktionszeit und Anzahl von Interaktionen im Netzwerk sowie eine Maximierung der Unabhängigkeit und Skalierbarkeit implementierter Systemkomponenten [Fiel00, S. 148]. Die Herausforderung verteilter Systeme, wie z. B. des WWW, besteht dabei aus der Integration einer Vielzahl von autonomen Komponenten, die auf unterschiedlichsten Plattformen implementiert sind [Fiel00, S. 66 ff.]. Zur Bewältigung dieser Herausforderung definiert REST sechs *Bedingungen* (REST-Constraints) anhand derer die Entwicklung von Architekturen verteilter AwS erfolgen soll [Fiel00, S. 76 ff.]. Im Fokus der REST-Bedingungen steht die Gestaltung von (Interaktions-)Beziehungen (engl.

[4] Unter einem Hypermedia-System wird ein netzwerkbasiertes System verstanden, in dem die Verknüpfung zwischen *Multimedia* (Elementen) verteilter Systemkomponenten durch *Hyperlinks* (Querverweise) realisiert werden [FeWa03, S. 589].

Connectors) zwischen den Komponenten des Systems, und nicht auf dem Entwurf der Komponenten selbst. Eine Architektur, die konform zu den Bedingungen gestaltet ist, wird als *RESTful* charakterisiert. Die Bedingungen sowie die damit verfolgten Ziele werden in den nachfolgenden Unterabschnitten vorgestellt.

3.2.1 Client-Server-Modell

Die erste REST-Bedingung ist die Einhaltung des *Client-Server-Modells* als grundlegende Voraussetzung für die Interaktion zwischen Komponenten der Web-Architektur [Fiel00, S. 78]. Der *Server* stellt seine Leistung als Dienst bereit und wartet auf Anfragen des Clients. Um einen Dienst zu konsumieren schickt der *Client* eine Anfrage an den Server. Dieser reagiert auf die Anfrage entweder mit deren Zurückweisung, oder er führt den aufgerufenen Dienst aus und sendet anschließend eine Antwort an den Client (z. B. [RaSc02, S. 177 f.], [Schm93, S. 38 f.]).

Die lose Kopplung zwischen Client und Server unterstützt deren unabhängige Weiterentwicklung. Zudem werden die Portierbarkeit der Nutzerschnittstelle und die Skalierbarkeit der Serverkomponente verbessert [Fiel00, S. 78].

3.2.2 Zustandslosigkeit

Die Kommunikation zwischen Client und Server ist grundsätzlich *zustandslos*. Dies bedeutet, dass jede Anfrage, die ein Client an einen Server stellt, alle relevanten Informationen beinhaltet, die zur Verarbeitung der Anfrage benötigt werden. Der Server selbst hält folglich keine transienten, clientspezifischen Daten des Anwendungszustandes über die Dauer der Anfragebearbeitung hinaus. Der Kontext einer Interaktion (Session-Zustand) liegt somit stets beim Client [Fiel00, S. 78 f.]. Es sei darauf hingewiesen, dass sich die Zustandslosigkeit auf die Kommunikation zwischen Systemkomponenten bezieht. Ressourcen- oder Anwendungszustände bleiben hiervon unberührt.

Durch die Einhaltung der zweiten Bedingung lassen sich Vorteile bezüglich Skalierbarkeit, Verlässlichkeit und Sichtbarkeit im Rahmen der Client-Server-Kommunikation erzielen, da der Server keine Systemressourcen für die Bereitstellung des Kontexts von aktuell laufenden Interaktionen aufwenden muss. Um die Performance-Nachteile einer zustandslosen und damit zwangsläufig redundanten Kommunikation zu bewältigen, wird die Cache-Bedingung eingeführt (z. B. [ECP+13, S. 54], [Fiel00, S. 79]).

Die Gestaltung von REST-Architekturen nach den ersten beiden Bedingungen setzt die Verwendung eines zustandslosen Client-Server-Protokolls voraus. Im WWW wird hierfür standardmäßig das *Hypertext Transfer Protocol* (*HTTP*) eingesetzt (Abschnitt 3.4.2).

3.2.3 Cache

Die *Cache-Bedingung* zielt auf die Verbesserung der Effizienz zustandsloser Kommunikation in verteilten Systemen. Sie fordert, dass die Antwortdaten eines Servers auf eine Client-Anfrage als „cacheable“ oder „non-cacheable“ gekennzeichnet werden können. Ist eine Antwort „cacheable“, so kann der Client nach festgelegten Kriterien auf ein wiederholtes Stellen

gleicher Anfragen verzichten und stattdessen auf seinen „gecachten“, lokal gespeicherten Datenbestand zugreifen [Fiel00, S. 79 f.].

Der Einsatz von „Caching“ erlaubt somit eine Reduktion der Anzahl an Interaktionen zwischen Client und Server. Die Bedingung ist, zusammen mit der Bedingung der Zustandslosigkeit, die zentrale Voraussetzung für die Verbesserung von Skalierbarkeit und Effizienz der Web-Architektur ([ECP+13, S. 55], [Fiel00, S. 80]).

3.2.4 Einheitliche Schnittstelle

Die einheitliche Schnittstelle, oder auch *REST-Schnittstelle*, fordert eine Verallgemeinerung (i. S. v. Vereinheitlichung) der Komponentenschnittstelle, die von einem Dienst bereitgestellt und durch einen Dienst-Konsumenten genutzt wird. Die REST-Schnittstelle ist die zentrale Bedingung zur konzeptionellen Gestaltung von Systemkomponenten und stellt das herausragende Unterscheidungsmerkmal von REST zu anderen Architekturstilen für netzbasierte Systeme dar. Mit der Einhaltung dieser Bedingung werden generell die Vereinfachung der Spezifikation von Systemschnittstellen und die Verbesserung der Sichtbarkeit von Interaktionen angestrebt. Die Implementierung der Operationen einer REST-Schnittstelle wird dabei durch die Komponenten des Systems verborgen, was die Entkopplung von Innen- und Außensicht und damit eine unabhängige Weiterentwicklung beider Systemsichten ermöglicht ([Mint05, S. 63], [Fiel00, S. 81 f.]).

Um eine anwendungsspezifische Interaktion zwischen Komponenten in verteilten Systemen zu realisieren, basiert die Anwendbarkeit des Konzepts der **einheitlichen Schnittstelle** wiederum auf der Einhaltung von insgesamt vier Bedingungen [Fiel00, S. 82].

- Jede Ressource des Systems ist *eindeutig identifizierbar*.
- Die Manipulation von Ressourcen erfolgt über deren *Repräsentation*.
- Die ausgetauschten Nachrichten zwischen Ressourcen sind *selbst-beschreibend*.
- Die Steuerung des Ausführungszustands einer Anwendung basiert auf dem Prinzip des „*hypermedia as an engine of application state*“.

Ressourcenorientierung und *eindeutige Identifikation* beschreiben in REST das grundlegende Konzept zur Gestaltung der Systemarchitektur. Als *Ressource* wird grundsätzlich jede Information bezeichnet, die anderen Systemteilnehmern zur Verfügung gestellt werden soll [Fiel00, S. 88 f.]. Dies können einfache Daten, wie Bilder, Videos und Texte, oder auch Dienste sein, die anwendungsspezifische Funktionalität bereitstellen. Jede Ressource besitzt einen *Identifikator* für ihre stabile und global eindeutige Identifikation im netzbasierten System [Niel99].

Die Implementierung der einheitlichen Schnittstelle basiert im WWW auf den Standards *Unified Resource Identifier* (URI) sowie *Hypertext Transfer Protocol* (HTTP). Ihr Einsatz bildet die technologische Grundlage für eine Realisierung der Identifikation und Manipulation von Ressourcen, der Spezifikation ausgetauschter Nachrichten sowie des Aufbaus einer Hypermedia. Verbreitete Repräsentationsformate zur Beschreibung von Ressourcenzuständen

sind z. B. die *JavaScript Object Notation* (JSON), *Extensible Markup Language* (XML) und *Hypertext Markup Language* (HTML). Eine Einführung dieser Standards findet sich in Abschnitt 3.4.

Eine ausführliche Vorstellung der einheitlichen Schnittstelle als zentrale Bedingung für die Gestaltung der Komponenten in RESTful AwS sowie der weiteren fünf REST-Bedingungen erfolgt in Abschnitt 3.3 im Zuge des Entwurfs von RESTful Services einer SOA.

3.2.5 Layered System

Die Bedingung des *Layered System* (Schichten-System) fordert die hierarchische Strukturierung einer REST-Architektur in Form eines mehrschichtigen Systems. Dabei kapselt jede Schicht die definierte Funktionalität einer Komponente und stellt diese über eine externe Schnittstelle bereit. Ruft z. B. ein Dienst-Konsument einen Dienst auf, so bleibt dem Konsumenten verborgen, welche weiteren Dienste im Zuge der Anfragebearbeitung genutzt werden [Fiel00, S. 82 ff.].

Mit Einhaltung der Forderung nach einem *Layered System* wird, neben dem Vorteil der Kapselung von Legacy-Systemen, insbesondere die transparente Weiterentwicklung eines Systems durch das Hinzufügen, Entfernen oder Modifizieren von Schichten verbessert. Ein Beispiel hierfür wäre die Einführung einer Middleware-Schicht zur Erhöhung der Skalierbarkeit von Diensten durch Load-Balancing ([Fiel00, S. 82 f.], [ECP+13, S. 56 f.]).

3.2.6 Code-On-Demand

Als letzte und einzige optionale Bedingung des REST-Architekturstils erlaubt *Code-On-Demand* eine Erweiterung der Anwendungslogik von Clients. Ist für die Realisierung einer zu erfüllenden Aufgabe erforderlich, dass der Client eine bestimmte Funktionalität bereitstellt, so kann er die fehlende Anwendungslogik vom Server herunterladen und ausführen oder, unabhängig vom Server, selber client-spezifische Funktionalität implementieren. Code-On-Demand wird in der Regel mit Technologien wie server-seitigen Applets oder client-seitigen Skriptsprachen realisiert [Fiel00, S. 84].

3.3 RESTful SOA

Zielsetzung der vorliegenden Arbeit ist die modellbasierte Spezifikation von Anwendungssystemen, die als RESTful SOA realisiert sind. Im Folgenden wird zunächst der Begriff RESTful SOA genauer untersucht. Anschließend werden die Eigenschaften von RESTful Services als zentrale Bausteine eingeführt (Abschnitte 3.3.1 und 3.3.2) sowie der softwaretechnische Aufbau von RESTful SOA konzipiert (Abschnitt 3.3.3). Die Ausführungen fokussieren dabei primär die Gestaltung der REST-Architektur aus dem Blickwinkel der Systementwicklung. Von technischen Aspekten der Implementierungsebene wird abstrahiert.

3.3.1 RESTful SOA und ROA

RESTful SOA ist eine service-orientierte Architektur, die konform zu den Bedingungen des Architekturstils REST ist und zur Unterstützung von Geschäftsprozessen dient. Das Verständnis von REST und SOA wird in der vorliegenden Arbeit somit im Kontext der Entwicklung betrieblicher AwS gesehen.

Verteilte, webbasierte und RESTful Systeme werden in der Literatur häufig auch mit dem Begriff *ressourcenorientierte Architektur*[5] (ROA) bezeichnet (z. B. [RiRu07, S. 79 ff.], [Over07, S. 341 ff.]). ROA wird als konkrete Architektur verstanden, die auf Basis ausgewählter Technologien und unter Einhaltung der REST-Bedingungen entwickelt wird, und sich aus einer Menge von Ressourcen als Systemkomponenten zusammensetzt. Eine Implementierung findet auf Basis der Technologien URI, HTTP sowie webbasierter Repräsentationsformate, z. B. XML oder JSON, statt [RiRu07, S. 79].
Der Begriff des (RESTful Web) Services wird im Umfeld der ROA unterschiedlich verwendet. Grundsätzlich ist jede Ressource auch ein Service und bildet damit einen elementaren Baustein der Architektur. Im weiteren Sinn stellt ein Service eine eigene Komponente dar, die ihre Funktionalität in Form einer Menge von Ressourcen veröffentlicht ([DVE10, S. 2 f.], [HdC09, S. 162 f.], [XZL+08a, S. 395], [LME08, S. 6 ff.], [Over07, S. 342 ff.]).

In der vorliegenden Arbeit wird die zu spezifizierende Zielarchitektur der Entwicklungsmethodik als *RESTful SOA* charakterisiert und demnach die fachliche Fokussierung ihrer Komponenten betont. In *Abgrenzung* zur ROA wird mit der RESTful SOA somit explizit die Unterstützung von Geschäftsprozessen in betrieblichen Systemen verfolgt.

3.3.2 RESTful Services

Dem Begriff des RESTful Service liegt in dieser Arbeit die allgemeine Definition von Services und SOA zugrunde, die um das Konzept der Ressourcenorientierung zur Realisierung der einheitlichen Schnittstelle erweitert wird (Abschnitte 3.1.2 und 3.2.4):

> *„Ein RESTful Service veröffentlicht eine Menge von Ressourcen, mit denen der Service-Nutzer interagiert, um die fachliche Funktionalität des Services zu nutzen.“*

Neben ihrer Konformität zu den REST-Bedingungen sind zwei Merkmale von RESTful Services besonders hervorzuheben. Zum einen stellt ein RESTful Service seine *Funktionalität als Menge von Ressourcen* mit einheitlicher Schnittstelle, und nicht in Form von anwendungsspezifischen Operationen bereit (z. B. [Paut14, S. 32 ff.], [ECP+13, S. 75], [Vino08a], [PZL08, S. 807], [RiRu07, S. 13]). Zum anderen wird die fachliche Ausrichtung des Services betont. Jeder RESTful Service kapselt eine definierte *fachliche Funktionalität* zur Unterstützung von Geschäftsprozessen.

RESTful Services sind anhand folgender zentraler **Merkmale** charakterisierbar (z. B. [Paut14, S. 32 ff.], [ECP+13, S. 68 ff.], [Tilk11, S. 11 ff.], [PZL08, S. 807]):

- Bereitstellung der Service-Funktionalität in Form von *Ressourcen mit einheitlicher Schnittstelle*
- *Zustandslose Client-Server-Kommunikation*

[5] Die Einführung und Definition der ROA als konkrete Architektur des REST-Stils lässt sich in der Literatur auf das Buch von RICHARDSON und RUBY aus dem Jahr 2007 zurückführen [RiRu07]. Im selben Jahr erfolgte auch die Veröffentlichung des Begriffs durch OVERDICK [Over07]. Viele Arbeiten aus dem REST-Umfeld nehmen Bezug auf diese zwei Beiträge (z. B. [LME08], [HdC09], [PZL08], [XZL+08a], [DVE10]). Die erste Verwendung des Begriffs *Resouce-oriented Architecture* (ROA) im Kontext verteilter Software-Architekturen konnte in einer Präsentation von THELIN [Thel03] aus dem Jahr 2003 sowie SNELL in 2004 [Snel04] gefunden werden (vgl. [RiRu07, S. 81]).

- Logische Verknüpfung in einer *Hypermedia*
- *Vertrag der RESTful Services* auf Basis der einheitlichen Schnittstelle

Die Merkmale von RESTful Services werden im Folgenden genauer erläutert.

3.3.2.1 Ressourcen mit einheitlicher Schnittstelle

Jeder RESTful Service veröffentlicht eine Menge von Ressourcen, über die seine Funktionalität genutzt wird. Ressourcen[6] bilden somit den elementaren Baustein einer RESTful SOA. Eine grundlegende Forderung ist dabei die Einhaltung der REST-Bedingungen der einheitlichen Schnittstelle (Abschnitt 3.2.4). Die drei wesentlichen Merkmale der einheitlichen Schnittstelle von Ressourcen sind die globale Adressierbarkeit, die Bereitstellung unterschiedlicher Repräsentationen und die Standard-Operationen [ECP+13, S. 68].

Mit dem generischen Konzept der Ressource wird in REST jede Abstraktion von Informationseinheiten bezeichnet, die über eine eindeutige Adresse zugreifbar sind. Die Art und Granularität der Information, die in einer Ressource gekapselt wird, ist dabei nicht definiert und konkretisiert sich erst aus dem Betrachtungsgegenstand und der Zielsetzung der Systementwicklung (z. B. [Tilk11, S. 11 f.], [WPR10, S. 4 f.], [Fiel00, S. 88 f.]).

EINDEUTIGER IDENTIFIKATOR

Jede Ressource verfügt über einen *stabilen und eindeutigen Identifikator* (ID), der sie global adressierbar und zugreifbar macht. Die Nutzung der Funktionalität eines RESTful Service erfolgt mittels Aufruf von Operationen über die ID seiner Ressourcen. In einer RESTful SOA werden IDs auf Basis von URIs spezifiziert (vgl. Abschnitt 3.4.1).

Die Gestaltung von URIs als globale Adresse von Ressourcen besitzt in REST-Architekturen, und im Speziellen im Kontext von RESTful SOA, eine große Bedeutung. Durch ihre Bezeichnungen und ihren hierarchischen Aufbau unterstützen URIs die Abbildung von Semantik und Abhängigkeiten der mit der Ressource unterstützten fachlichen Anforderungen. Als Grundregeln für den Identifikatorenentwurf gelten insbesondere (z. B. [Tilk11, S. 44 f.], [Alla10, S. 75 ff.], [Niel99]):

- Verwendung *sprechender Bezeichner*. Eine Ressource ist ein Gegenstand und keine Operation. Sie wird mit einem Substantiv benannt.
- *Hierarchischer Aufbau* zur Darstellung von existenziellen Abhängigkeiten und Bildung von Gruppen fachlich zusammengehöriger Ressourcen.

Beispiele für die Gestaltung von URIs werden im nachfolgenden Abschnitt zu Ressourcentypen gegeben. Für Ressourcen, die eine Menge von Ressourceninstanzen umfassen, wird dabei

[6] Als Ressource wird in der REST-Literatur ein abstraktes Konzept verstanden. Nach RICHARDSON und RUBY ist eine Ressource grundsätzlich *„jedes Ding, das bedeutend genug ist, um referenziert zu werden“* ([RiRu07, S. 81], eigene Übersetzung). OVERDICK versteht hierunter jede Entität, die mit einem Nomen benannt werden kann [Over07, S. 341]. Übertragen auf das WWW können Ressourcen beispielsweise Webseiten, Bilder, Videos, Dokumente oder Web-Dienste sein. Diese „Objekte“ besitzen eine URI als globale Adresse und sind über HTTP-Methoden zugreifbar.

standardmäßig der Plural verwendet. Zur systematischen Bestimmung von IDs ist ein methodisches Vorgehen innerhalb des angewendeten Entwicklungsprozesses erforderlich.

RESSOURCENTYPEN

Zur besseren Strukturierung des Konzepts der Ressource werden diese anhand der Art, der durch sie bereitgestellten Informationen in Typen untergliedert. Wichtige Ressourcentypen zur Realisierung von RESTful Services sind (nach [Tilk11, S. 34 ff.]):

- *Primärressource (Individualressource)*: Im Kontext von betrieblichen AwS referenziert eine Primärressource diejenigen Informationseinheiten, die Kandidaten für eine Persistierung sind. Es handelt sich somit meist um die (persistierten) Instanzen eines Kernkonzepts der Domäne, weshalb auch der Begriff der Individualressource verwendet wird.
  Beispiel hierfür sind die Kunden eines Systems, die über die URI der Kundenressource `/customers/{id}` identifiziert werden. Mit `{id}` wird der Platzhalter für eine konkrete Kundennummer angegeben.
- *Subressource*: Steht eine Ressource mit einer anderen Ressource in einer Aggregations- oder Generalisierungsbeziehung, so wird sie als Subressource bezeichnet.
  Beispielsweise sind Auftragspositionen stets Teil von existierenden Aufträgen. Die URI zur Liste der Positionen eines Auftrags könnte z. B. `/orders/{id}/positions` lauten.
- *Listenressource*: In einer Listenressource können die Individualressourcen, und damit die Instanzen, eines Kernkonzepts der Domäne aufgelistet werden.
  Ein Beispiel hierfür ist die Liste aller Kunden mit der URI `/customers`. Der Inhalt der Kundenliste könnten die Referenzen auf alle angelegten Kunden sein.
- *Filterressource*: Eine Listenressource, die eine Individualressource nach einem definierten Kriterium selektiert und auflistet, wird als Filterressource bezeichnet (Symbol ist `?`).
  Als Beispiel kann über `/customers?status={status}` auf eine Liste mit allen Kunden zugegriffen werden, die einen bestimmten Status besitzen.
- *Aktivitätsressource*: Aktivitätsressourcen bieten eine bestimmte Funktionalität (Aktivität) im Rahmen der Durchführung von Workflows an.
  Die Verarbeitungslogik, die bei der Durchführung des Vorgangs „Auftragsprüfung“ im Workflow „Auftragsabwicklung“ gestartet wird, wäre dafür ein Beispiel.

Darüber hinaus ist die Bildung weiterer Kategorien von Ressourcentypen möglich ([Tilk11, S. 35 ff.], [Schr11, S. 17 f.]). Diese spielen jedoch bei der Entwicklung von RESTful SOA in dieser Arbeit eine untergeordnete Rolle und werden nicht näher.

REPRÄSENTATION

Für die Interaktion mit anderen Systemkomponenten stellen Ressourcen eine oder mehrere Repräsentationen bereit. Eine *Repräsentation* beschreibt einen definierten Zustand der Ressource in einem festgelegten Dokumentformat. Der Client greift über den Identifikator einer Ressource auf ihre Repräsentation zu und bestimmt das konkrete Austauschformat mittels *Content Negotiation* [Tilk11, S. 83]. Durch die Trennung von Repräsentation und

Ressource ist eine flexible Bestimmung des Formats der übermittelten Zustandsinformationen möglich ([Fiel00, S. 90 f.], [RiRu07, S. 91 f.], [WPR10, S. 7 ff.]).

In webbasierten Systemen wird das Datenformat von Repräsentationen über den *Medientyp* charakterisiert (Abschnitt 3.4.3). Zum Beispiel kann die Ressource „Auftrag“ ihren Zustand über Repräsentationen in den Formaten XML und JSON bereitstellen. Ein Client, der die Daten eines Auftrags abrufen möchte, greift über die URI auf die Ressource zu und bestimmt die akzeptablen Formate, in denen die Repräsentation des Auftragszustands übermittelt werden soll.

Der dokumentenbasierte Austausch von Repräsentationen zwischen den Komponenten der RESTful SOA ist der zentrale Ansatz, um die Anwendungsausführung bzw. den Ablauf eines Geschäftsvorfalls zu steuern. Die Manipulation von Zustandsinformationen des AwS basiert auf der Verarbeitung und Persistierung von Dokumenten. Die bedeutende Rolle von Dokumenten in der RESTful SOA und die zielgerichtete Gestaltung von Ressourcen-Repräsentationen bzw. die Spezifikation der dazu korrespondierenden Dokumenttypen wird mit dem Begriff der *Dokumentenorientierung* erfasst [WoBe13, S. 1230].

STANDARD-OPERATIONEN

Die öffentliche Schnittstelle der Ressourcen einer RESTful SOA umfasst eine Menge von Standard-Operationen (HTTP-Methoden[7]) mit definierter Semantik [ECP+13, S. 72 f.]. Durch den Einsatz von HTTP sind die Menge verfügbarer Operationen, ihre Bedeutung und Wirkungsweise, ihre Eigenschaften sowie der grundlegende Aufbau von Anfrage- und auch Antwortnachrichten zwischen Client und Server festgelegt (Abschnitt 3.4.2).

3.3.2.2 Service-Vertrag

Die Bereitstellung einer einheitlichen Schnittstelle erlaubt einem Client mit einer Ressource in Interaktion zu treten, sobald er die Bedeutung des Gegenstands und die URI dieser Ressource kennt. Die Semantik der verfügbaren Operationen und der Aufbau der Nachrichten sind im HTTP-Standard definiert. Zudem wird das Datenformat der ausgetauschten Repräsentationen während der Interaktion verhandelt (Content Negotiation).

Die Funktionalität eines RESTful Services wird in seinem Vertrag spezifiziert und über diesen öffentlich zugänglich gemacht. Ein *Service-Vertrag* umfasst eine Menge von Ressourcen, die anhand ihrer URI, HTTP-Methoden und der unterstützten Medientypen beschrieben werden [ECP+13, S. 75 f.].

Abbildung 3.3 visualisiert das Beispiel eines RESTful Service-Vertrags zur Verwaltung von Kunden. Die Ressourcen des Entitätsservices *Kunde* sind in Form weißer Rechtecke mit abgerundeten Ecken als Symbol dargestellt. Im Beispiel stellt der RESTful Service seine Funktionalität mittels dreier Ressourcen zur Verfügung. Zur Nutzung des Services fragt ein Client z. B. den Kunden mit der Nummer 815 an und möchte die Repräsentation der aktuellen

[7] Im Zusammenhang mit der einheitlichen Schnittstelle wird meist von den „Operationen“ eines Services gesprochen. In dieser Arbeit werden die Begriffe (HTTP-)*Operation* und (HTTP-)*Methode* synonym verwendet, da GET, POST, PUT, etc. in der HTTP-Spezifikation als *Methoden* bezeichnet werden [FGM+99]. In der Literatur findet sich zudem der Begriff (HTTP-)*Verb* (z. B. [WPR10, S. 11], [Tilk11, S. 51]).

Kundendaten als JSON-Dokument erhalten. Hierzu nutzt er die HTTP-Methode *GET*, die der Service implementiert und bereitstellt.

```
GET /customers/815  HTTP/1.1
Host: simple.customer.example.de
Accept: application/json
```

Die Antwortnachricht des Service umfasst im Kern den Verarbeitungsstatus in Form eines *Response Code* (erfolgreich: *200 OK*), den Medientyp des Nachrichtenkörpers (*Content-Type*) sowie die aktuelle Repräsentation des abgefragten Ressourcenzustands.

```
200 OK
Content-Type: application/json
{ customerID : 815, ... }
```

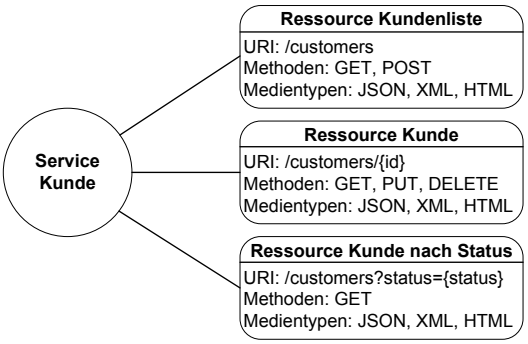


Abbildung 3.3: Vertrag eines RESTful Services *Kunde*

Der Client empfängt die HTTP-Antwort-Nachricht des Services als Ergebnis der Anfrageverarbeitung. Die Implementierung der öffentlichen GET-Methode bleibt dem Client verborgen. Der Einsatz von HTTP als Anwendungsprotokoll, sowie Medientypen und URI, ermöglichen damit die Durchführung und Verarbeitung von Service-Aufrufen unabhängig von der Bereitstellung weiterer Implementierungsdetails. Weitere Beispiele, die eine Nutzung der Schnittstelle von RESTful Services beschreiben, finden sich z. B. in [Tilk11, S. 19 ff.] und [ECP13, S. 75 ff.].

Anwendungsspezifische Operationen

Der Unterschied zwischen der Gestaltung einer RESTful und einer „klassischen" Schnittstelle, welche die Realisierung in Form einer Menge *anwendungsspezifischer Operationen* vorsieht, wird nachfolgend dargelegt. Die Schnittstelle des Service *Kunde* aus dem vorausgegangenen Beispiel (Abbildung 3.3) wird dazu durch eine Menge anwendungsspezifischer Operationen beschrieben (z. B. [ECP+13, S. 75 f.], [Tilk07, S. 399]). Abbildung 3.4 stellt diese als weißes Rechteck mit rechtwinkligen Ecken dar.

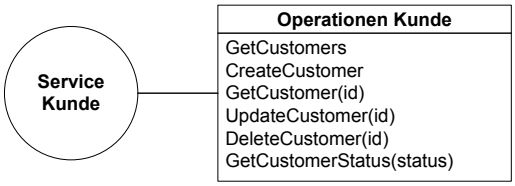


Abbildung 3.4: Anwendungsspezifische Schnittstelle des Service *Kunde* (in Anlehnung an [Tilk07, S. 399])

Zur Nutzung dieser anwendungsspezifischen Schnittstelle müssen insbesondere die Aufrufparameter sowie die Struktur der Anfrage- und Antwort-Nachrichten für die einzelnen Operationen definiert werden. Beispielsweise sind die Bedeutung der Operation *CreateCustomer* sowie die Struktur der zugehörigen Aufruf-Nachricht nicht bekannt. Die Spezifikation von Semantik und Nutzungsinformationen der veröffentlichten Operationen erfolgt i. d. R. in einem eigenen Dokument (z. B. WSDL [CMR+07]).

Ein weit verbreiteter Ansatz zur Realisierung anwendungsspezifischer Service-Schnittstellen beruht auf dem Einsatz von Web-Service-Technologien (Abschnitt 3.1.1). Die Diskussion alternativer Ansätze zur Gestaltung von Services oder ein Vergleich der hierbei eingesetzten Technologien ist nicht Gegenstand der vorliegenden Arbeit. An dieser Stelle sei lediglich auf weiterführende Literatur zu diesem Thema verwiesen (z. B. [ECP+13, S. 77 ff.], [Tilk11, S. 207 ff.], [PZL08], [Vino07], [Mint05], [Balz11, S. 318]).

3.3.2.3 Zustandslose Kommunikation

Das Prinzip der zustandslosen Kommunikation fordert, dass ein RESTful Service im Rahmen der Kommunikation mit einem Client keine Informationen der Interaktion (Sessionzustand) über die Dauer der Verarbeitung von Anfragen hinaus verfügbar hält. Das Prinzip ist das Ergebnis der Anwendung der Bedingung „Zustandslosigkeit“ bei der Gestaltung von Services (Abschnitt 3.2.2). In der Konsequenz ergeben sich zwei Anforderungen an die Durchführung der Kommunikation ([Fiel00, S. 78 f. und S. 121 f.], [RiRu07, S. 86 ff.], [Tilk14, S. 8]):

- Der Zustand einer Client-Service-Interaktion wird entweder vom Service in Form eines *Ressourcenzustands* persistiert, oder als Clientzustand gehalten.
- Die ausgetauschten Nachrichten sind *selbst-beschreibend* (*self-descriptive messages*) und enthalten alle Informationen die benötigt werden, um eine Nachricht zu verstehen und verarbeiten zu können.

Das Ziel der zustandslosen Kommunikation ist es, die Skalierbarkeit verteilter Systeme zu erhöhen und die Kopplung ihrer Komponenten zu verringern [Fiel00, S. 78 f.]. Sie wird in einer RESTful SOA, oder allgemein in ROA, durch den Einsatz von HTTP zur Spezifikation von Anfrage- und Antwortnachrichten realisiert. Im Nachrichtenkopf können Informationen über die Nachricht an sich, sowie den Inhalt ihres Körpers übermittelt werden. Metainformationen sind z. B. der Content-Type und Response Code einer Antwort oder die URIs weiterer Ressourcen im gesendeten Nachrichtenkörper ([Fiel00, S. 121 ff.], [Paut14, S. 33], [RiRu07, S. 86], [Tilk14, S. 8]).

3.3.2.4 Hypermedia

In REST-Architekturen werden Beziehungen zwischen Ressourcen in Form von Hyperlinks spezifiziert. Diese spannen eine *Hypermedia* auf. Macht ein Server dem Client im Rahmen einer Anfrageverarbeitung z. B. relevante Verknüpfungen über die gesendete Repräsentation einer Ressource verfügbar, so wird damit die Steuerung des Ausführungsablaufs innerhalb des Systems ermöglicht. Repräsentationen beschreiben aber nicht nur Ressourcenzustände, sondern auch Beziehungen zu anderen Ressourcen (z. B. [Tilk11, S. 12 f.], [PWS+10, S. 16 f.], [Vino08a, S. 94], [RiRu07, S. 94 ff.]).

Die Realisierung der Ablaufsteuerung über Verknüpfungen wird von FIELDING als REST-Bedingung unter der Bezeichnung „*hypermedia as an engine of application state*“ eingeführt (Abschnitt 3.2.4, [Fiel00, S. 82]). In der Literatur wird häufig auch die Abkürzung *HATEOAS* oder einfach *Hypermedia Constraint* verwendet (z. B. [Paut14, S. 35], [Burk13, S. 139 ff.], [PWS+10, S. 16], [AWB11, S. 112], [Vino08a, S. 94]).

Die Hypermedia dient in einer RESTful SOA vor allem der Spezifikation von Beziehungen zwischen den Ressourcen von RESTful Services. Ein Service stellt hierzu in seinen Antwortnachrichten Verknüpfungen auf weitere Ressourcen desselben und/oder anderer Services bereit. Der Client greift auf die verlinkten Ressourcen über die erhaltene URI zu und konsumiert diese über die angebotenen Standard-Operationen. Die bereitgestellten Links können einerseits *strukturelle Beziehungen* zwischen Ressourcen, oder andererseits die *Ablaufbeziehungen* in einem Vorgangsnetz beschreiben. Als Startpunkt für die Nutzung der Funktionalität einer RESTful SOA können zudem *Einstiegspunkte* erstellt werden, die auf eine Menge von Ressourcen verlinken ([ECP+13, S. 83 ff.], [Tilk11, S. 74 ff.]).

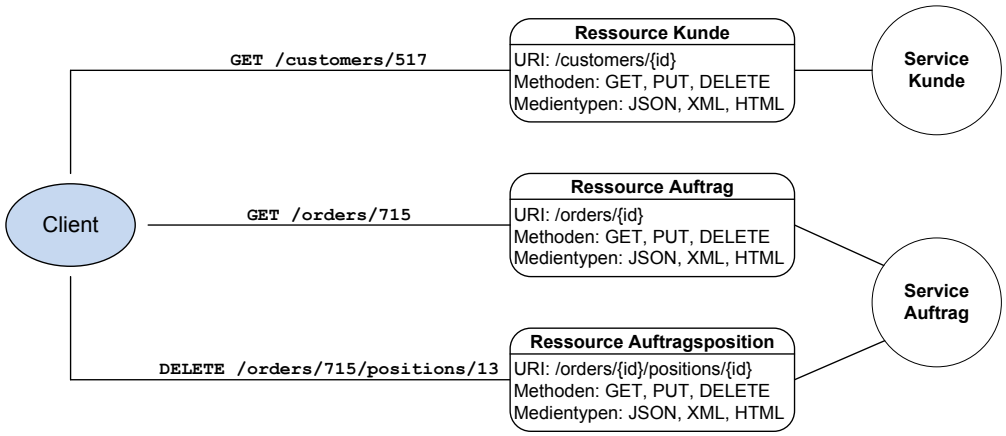


Abbildung 3.5: Nutzung von Hyperlinks am Beispiel einer Auftragsverwaltung

Abbildung 3.5 zeigt das Prinzip der Hypermedia am Beispiel einer Auftragsverwaltung. Der Client fragt die Daten des Auftrags 715 beim Service *Auftrag* an (`GET`). Dieser übermittelt in seiner Antwort, neben den Auftragsdaten selbst, Links auf die einzelnen Auftragspositionen sowie den Kunden als Auftragsersteller. Diese Informationen können einem menschlichen bzw. maschinellen Nutzer z. B. als HTML- bzw. JSON-Dokument zur Verfügung gestellt werden. Auf Basis der Auftragsdaten löscht der Client nun die Auftragsposition 13 mittels `DELETE` auf die entsprechende URI. Des Weiteren werden über die bereitgestellte Referenz aus der Auftragsrepräsentation die Daten des Kunden lesend aufgerufen, jedoch nicht weiter manipuliert.

3.3.2.5 Weitere REST-Bedingungen

Neben den eingeführten zentralen Merkmalen von RESTful Services sind noch *Caching* (*Cache*) und *Layered-System* zu berücksichtigen, um Services nach den eingeführten Bedingungen „vollständig“ REST-konform zu gestalten (Abschnitt 3.2).

CACHE

Der Einsatz von Caches stammt ursprünglich aus dem Umfeld des WWW und dient dem lokalen Verfügbarmachen seiner Ressourcen (z. B. Bilder, HTML-Seiten etc.). Das vorrangige Ziel von Caching ist hierbei die Verbesserung der Skalierbarkeit in netzbasierten Systemen durch das lokale (clientseitige) Speichern der Ergebnisse von Service-Aufrufen. Hierzu wird bei der wiederholten Abfrage von Repräsentationen zunächst der lokale Cache nach Kopien des anzufragenden Ressourcenzustandes durchsucht, um diese auf Gültigkeit zu überprüfen. Die Prüfung erfolgt entweder anhand der Gültigkeitsdauer oder der Validierung beim Server. Durch die Möglichkeit der Verwendung von lokalen Kopien lassen sich die im Netzwerk übertragene Datenmenge sowie die Anzahl der durchgeführten Serverzugriffe verringern (z. B. [Tilk11, S. 115 ff.], [Alla10, S. 147 ff.], [WPR10, S. 157 ff.], [Fiel00, S. 79 ff.]).

Bei der Gestaltung von RESTful SOA spielt die Cache-Bedingung für die Systementwicklung in der vorliegenden Arbeit nur eine untergeordnete Rolle. Dies hat insbesondere zwei Gründe:

- In RESTful SOA können grundsätzlich die potenziell geschäftskritischen Inhalte von Ressourcen gegen den Einsatz von Caches sprechen. Zum einen ist es wahrscheinlich, dass die lokale Speicherung von geschäftlichen oder persönlichen Daten (bei einem externen Client) nicht gewünscht ist. Zum anderen ist in vielen Fällen die Gefahr einer Inkonsistenz zwischen lokalen und serverseitigen Daten nicht tolerierbar. In der Konsequenz kann dies ein häufiges Validieren und Nachladen der Daten und damit eine Erhöhung des Verarbeitungsaufwands zur Folge haben, der den Vorteilen entgegenwirkt [WPR10, S. 159 f.].
- Beim Caching handelt es sich um eine technologische Anforderung, die aus konzeptueller Sicht keinen direkten Einfluss auf die modellbasierte Spezifikation der RESTful Services und der Service-Architektur hat. Ihre Umsetzung findet primär auf der Implementierungsebene Berücksichtigung und ist dort Teil der Realisierung fachlich begründeter Service-Aufrufe.

Der Fokus liegt in dieser Arbeit auf der softwaretechnischen Gestaltung der RESTful Services auf Basis der fachlichen Anforderungen in SOM-Geschäftsprozessmodellen.

LAYERED SYSTEM

Ziel der Bedingung *Layered System* ist die Anpassbarkeit von Systemen durch z. B. das Hinzufügen oder Entfernen von Schichten zu fördern und die Komplexität der Infrastruktur zu verbergen [Fiel00, S. 82 ff.]. Eine wichtige Voraussetzung zur Umsetzung der Schichten-Bedingung ist die Verwendung selbst-beschreibender Nachrichten, damit eine globale Weiterleitung von Anfragen und Antworten im Rahmen von Serviceaufrufen erfolgen kann. Darüber hinaus sollten alle Komponenten des Systems ein gemeinsames Kommunikationsprotokoll verwenden ([ECP+13, S. 204], [Fiel00, S. 82 ff.]).

Das Schichten-System wird im Sinne des Architekturstils REST somit primär unter dem technologischen Gesichtspunkt der zugrundeliegenden Infrastruktur gesehen. Von Aspekten einer hardwaretechnischen Realisierung zur Erfüllung dieser Bedingung wird in dieser Arbeit abstrahiert. Dagegen spielt die Schichtenbildung nach fachlichen Kriterien beim Entwurf von RESTful SOA eine wichtige Rolle für die Gestaltung der REST-Architektur.

3.3.3 Aufbau von RESTful SOA

Ausgehend vom vorgestellten Verständnis der SOA sowie der RESTful Services und Ressourcen wird abschließend die Gestaltung von RESTful SOA als konkrete service-orientierte Architektur genauer untersucht. Die Rollen und Serviceklassen von SOA werden bereits in den Abschnitten 3.1.1 und 3.1.4 vorgestellt. Nachfolgend werden das *Schichtenmodell* sowie, darauf aufbauend, die *Softwarearchitektur von RESTful SOA* schrittweise erarbeitet und verfeinert.

3.3.3.1 Schichtenmodell der RESTful SOA

CLIENT-SERVER-MODELL

Die Strukturierung von RESTful SOA basiert grundlegend auf dem *Client-Server-Prinzip* und der Differenzierung nach Service-Konsument (Client) und Service-Anbieter (Server) (Abschnitte 3.1.1 und 3.2.1). Die Service-Anbieter stellen ihre Funktionalität als Menge von Diensten bereit, welche ihrerseits eine Menge von Ressourcen veröffentlichen, die ein Service-Konsument im Zuge der Client-Server-Kommunikation nutzt.

SERVICEKLASSEN

Auf Seiten des Service-Anbieters wird die RESTful SOA nach dem Kriterium der Serviceklasse strukturiert. In service-orientierten Architekturen wird hierbei nach der Art der angebotenen Funktionalität zwischen *Entitäts-* und *Vorgangsdiensten* zur Unterstützung der Geschäftsprozessdurchführung differenziert. Die Vorgangsdienste werden wiederum in *nicht-elementare* sowie *elementare Dienste* untergliedert (Abschnitt 3.1.4).

Die Menge aller Dienste einer Klasse bildet jeweils eine *Schicht* für die *entitätsspezifische Geschäftslogik* und die *Ablauflogik*. Die Ablaufschicht selbst wird anhand des Merkmals der Realisierung von Lösungsverfahren elementarer Vorgänge oder von Workflows zur Durchführung (nicht-elementarer) Vorgangsnetze in zwei Schichten unterteilt.

SCHNITTSTELLEN FÜR DIE BENUTZERINTERAKTION

Die Realisierung der Client-Server-Kommunikation erfordert die Bereitstellung nutzerkonformer Schnittstellen. In der SOA-Literatur werden Benutzerschnittstellen häufig in Form spezieller Komponenten (Application Frontends) zur MC- oder CC-Kommunikation realisiert und in einer eigenen Schicht bereitgestellt (z. B. [Tilk11, S. 191 ff.], [Josu08, S. 135 ff.], [KBS05, S. 69], [Arsa04]). Die Interaktion mit externen Benutzern ist darüber hinaus nach dem Merkmal der Kooperation mit personellen oder maschinellen Aufgabenträgern (K_P und K_M) unterscheidbar (Abschnitt 2.2.4.3). Gegenstand der Entwicklung von Benutzerschnittstellen ist die Gestaltung der Nutzeroberfläche (z. B. GUI oder AwS-Schnittstelle) sowie die Festlegung möglicher Kommunikationsprotokolle, die für eine Realisierung der CC- oder MC-Kommunikation von SOA, sowie K_P oder K_M erforderlich sind.

Durch ihre Konformität zu den REST-Bedingungen und die Verwendung von HTTP ist die Interaktion zwischen Komponenten von RESTful SOA bereits zu einem hohen Grad standardisiert und ihre technische Realisierung festgelegt. Der Zugriff auf die RESTful Services über die Benutzerschnittstelle ist jedoch für K_P oder K_M noch konkret zu spezifizieren. Die Art

der Kommunikation, die Details des Zugriffs sowie der (Nutzer-)Oberfläche werden deshalb in der Komponente *Interface-Objekt* gekapselt.

SCHICHTENMODELL

Den Aufbau der RESTful SOA visualisiert Abbildung 3.6 als Schichtenmodell in konzeptueller Form. Die Betrachtung der Aufgabenträgerebene erfolgt dabei aus der Außensicht. Die grundlegende hierarchische Gliederung von SOA findet sich in einer Vielzahl von Ansätzen wieder (z. B. [FeSi13, S. 495], [KrSi11, S. 293 f.], [Erl08b, S. 43 f.], [Sied07], [AZE+07], [KBS05, S. 82 ff.]).

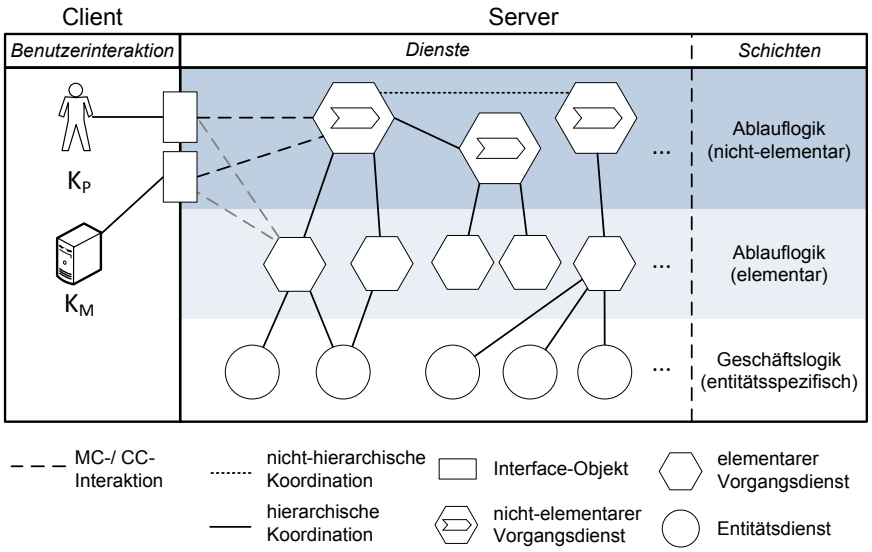


Abbildung 3.6: Schichtenmodell der RESTful SOA

Die *Benutzerinteraktion* von Client und Server erfolgt über Interface-Objekte zwischen den Nutzern K_P und K_M sowie den RESTful Services. Die Dienste werden dabei hierarchisch in Schichten gegliedert. Die *nicht-elementaren Vorgangsdienste* (neVD) realisieren ihre Funktionalität durch die Orchestrierung einer Menge von Diensten. Zudem können sie in einer Choreographie-Beziehung mit weiteren nicht-elementaren Vorgangsdiensten stehen. *Elementare Vorgangsdienste* (eVD) greifen wiederum auf eine Menge von *Entitätsdiensten* (ED) zu (nach Abschnitt 3.1.4).

Eine RESTful Gestaltung von SOA unterstützt die Auslagerung von Teilen der Ablaufsteuerung auf die Benutzerebene. Zentrale Konzepte zur Förderung dieser Auslagerung sind die Bereitstellung einer Hypermedia sowie der einheitlichen Schnittstelle. Beide Konzepte ermöglichen die koordinierte Erschließung und direkte Nutzung von elementaren Services der RESTful SOA (z. B. [Alla10, S. 251 ff.], [Vino08b]). Hinsichtlich der Realisierung von RESTful SOA bestehen somit *Freiheitsgrade* bezüglich einer konkreten Gestaltung der Ablaufsteuerung. Diese Alternative ist in der abgebildeten Schichtenarchitektur angedeutet (grau gestrichelte Linien).

3.3.3.2 Softwarearchitektur der RESTful SOA

Ausgangspunkt einer Konkretisierung der softwaretechnischen Architektur der RESTful SOA ist ihr konzeptueller Aufbau als Schichtenmodell. In der bisherigen Untersuchung wurde der Aufbau aus der Außenperspektive betrachtet und von einer softwaretechnischen Realisierung

abstrahiert. Im Folgenden wird der Systemaufbau aus Außensicht in Bezug auf die Umsetzung der RESTful Dienste konkretisiert und um die innere Softwarestruktur erweitert. Dies führt zu einer Strukturierung in die Ebenen Ressourcen, Dienste, Software und Datenhaltung sowie die Nutzerebene.

RESSOURCEN UND DIENSTE

Aus Außensicht stellt die RESTful SOA ihre Funktionalität als Menge von RESTful Services bereit. Jeder Dienst spezifiziert seine Schnittstelle wiederum als Menge von Ressourcen, die gemäß ihrer fachlichen Ausrichtung in die Kategorien *Vorgangsressource* (VR) oder *Entitätsressource* (ER) eingeteilt werden. Jeder Entitäts- bzw. Vorgangsdienst wird demnach durch eine Menge von Entitäts- bzw. Vorgangsressourcen erbracht. Sowohl die Kommunikation mit externen Nutzern als auch zwischen den Komponenten der SOA selbst erfolgt über die einheitliche Schnittstelle der Ressourcen. Eine Implementierung basiert standardmäßig auf dem Einsatz der Standards HTTP und URI (Abschnitte 3.1.4 und 3.3.2).

SOFTWARE

Das anwendungsspezifische Lösungsverfahren von RESTful Services wird aus der Innenperspektive durch *Softwarekomponenten* realisiert. Dabei wird die Implementierung der Dienste gekapselt und „hinter“ ihren Schnittstellen vor den Service-Nutzern verborgen ([Sied07, S. 111 f.], [KBS05, S. 82 ff.], [Arsa04], [EAA+04, S. 23 f.]).

Der Vorteil einer Trennung von Diensten und Softwarekomponenten ist, dass mehrere Dienste, die im Rahmen ihrer Durchführung auf überlappende Datenstrukturen zugreifen, von einer Softwarekomponente erbracht werden können. Übertragen auf das Modell der betrieblichen Aufgabe führen Dienste somit Aufgaben durch, die ein gemeinsames Aufgabenobjekt besitzen ([KrSi11, S. 293 ff.] und Abschnitt 2.1.3).

DATENHALTUNG

Zur Verwaltung persistenter Daten greifen Softwarekomponenten auf die Komponenten der Datenhaltungsebene zu. *Datenhaltungskomponenten* dienen der Manipulation und Speicherung der Zustände von Entitätsressourcen (z. B. [Josu08, S. 135 ff.], [Tilk11, S. 191 ff.]). Die Datenhaltung ist dabei zunächst unabhängig von der Wahl eines bestimmten Datenbankverwaltungssystems oder Datenmodells. Eine Speicherung der Zustände von Vorgangsressourcen ist grundsätzlich nicht vorgesehen, da das Ergebnis ihrer Ausführung/ihres Aufrufs in Form der Zustände von den verwalteten Entitäten repräsentiert wird[8]. Der Ansatz einer Trennung von Vorgängen und Entitäten unterstützt das Prinzip der zustandslosen Kommunikation in RESTful SOA.

SOFTWARETECHNISCHER AUFBAU DER RESTFUL SOA

Abbildung 3.7 stellt den softwaretechnischen Aufbau der RESTful SOA dar. Die Aufgabenträgerebene des Systems ist hierarchisch strukturiert. Aus der Außensicht ergibt sich eine

[8] Aufgrund technischer Anforderungen kann jedoch eine Speicherung von Vorgangszuständen notwendig werden, um sie beispielsweise als langlaufende Workflows zu realisieren.

Gliederung in die Ebenen der *Ressourcen* (hellblau) und der *Dienste* (mittelblau). Durch die Nutzerebene erfolgt der (interne oder externe) Zugriff auf die veröffentlichten Ressourcen des Systems. Aus der Innensicht (dunkelblau) wird die Funktionalität von RESTful Services durch *Softwarekomponenten* implementiert (*Softwareebene*). Zur Realisierung der Datenverwaltung wird auf Datenhaltungskomponenten zugegriffen (*Datenhaltungsebene*).

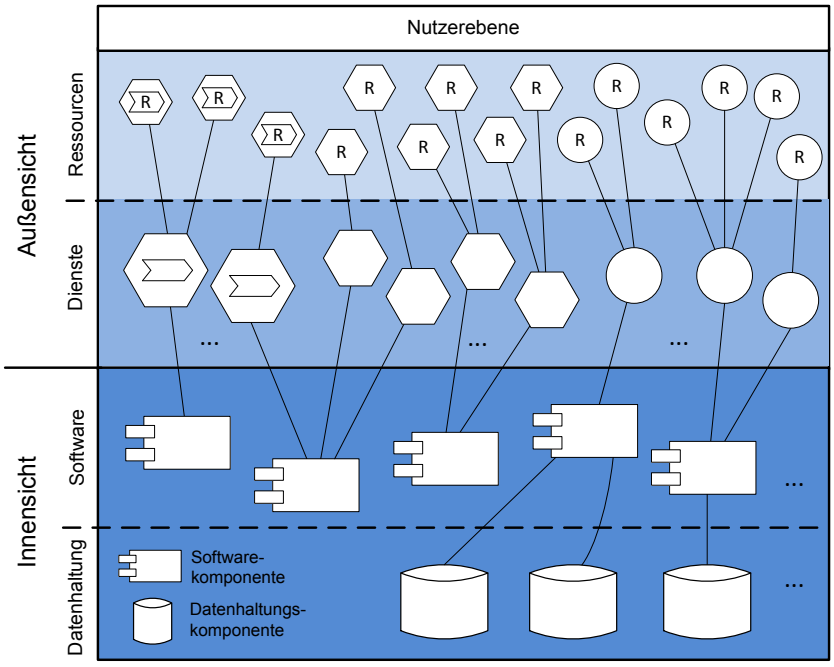


Abbildung 3.7: Softwaretechnischer Aufbau der RESTful SOA (Innen- und Außensicht)

Die vorliegende Abbildung zielt auf die Darstellung der softwaretechnischen Struktur von RESTful SOA. Dabei wird nicht zwischen der Interaktion mit internen oder externen Nutzern unterschieden, welche die *Nutzerebene* kapselt. Ebenso wird von einer hierarchischen Anordnung der Dienste und Ressourcen abstrahiert. Hierfür gilt der Aufbau nach Abbildung 3.6.

3.4 Exkurs: Technologien zur Realisierung von REST-Anwendungen

Die Realisierung von RESTful AwS sowie der Interaktionsbeziehungen zwischen ihren verteilten Systemkomponenten fußt wesentlich auf dem Konzept der einheitlichen Schnittstelle (Abschnitt 3.3.2.1). Um die technische Umsetzung der Anforderungen von Aufbau und Nutzung der REST-Schnittstelle näher zu beleuchten, werden in dem nachfolgenden Exkurs Standard-Technologien vorgestellt, die bei der Implementierung der REST-Bedingungen zum Einsatz kommen. Dies sind:

- Die Syntax zur eindeutigen Identifikation von Ressourcen und, darauf aufbauend, der Konstruktion einer Hypermedia.
- Das Anwendungsprotokoll für die Implementierung von Standard-Operationen der Ressourcen.
- Die Medientypen zur Repräsentation der Daten von Ressourcen.

Die Gestaltung der REST-Architektur ist zunächst unabhängig vom Einsatz konkreter Technologien. In Wissenschaft und Praxis sind jedoch die Spezifikationen von Uniform Resource Identifier, Hypertext Transfer Protocol und Internet Media Types (MIME) als „Quasi"-Standards von RESTful Systemen etabliert (z. B. [ECP+13, S. 69 ff.], [Tilk11, S. 9 ff.], [PZL08, S. 807], [Wild07, S. 2 f.]). Als Plattform des WWW besitzen sie eine sehr hohe Verbreitung und stellen aktuell die einzig bedeutenden Technologien für die Umsetzung der REST-Schnittstelle und Bereitstellung eines Anwendungsprotokolls für die zustandslose Kommunikation dar. Der Einsatz alternativer Basismaschinen ist zwar theoretisch möglich, besitzt in der Praxis jedoch keine größere Relevanz.

3.4.1 Uniform Resource Identifier

Der Standard *Uniform Resource Identifier* (URI) [BFM05] definiert die Syntax zur Spezifikation der Identifikatoren von Ressourcen. Die URI-Syntax legt hierfür die generische Struktur des Identifikators, die Menge zulässiger und verbotener Zeichen sowie die Teilmengen *Uniform Resource Locator* (URL) und *Uniform Resource Name* (URN) als verwandte Schemata fest [BFM05, S. 6 f.]. Die Verwaltung des URI-Standards obliegt der Internet Engineering Task Force (IETF)[9].

Die *generische URI-Syntax* definiert der aktuelle Standard RFC 3986 [BFM05, S. 16]:

```
{scheme}"://"[{authority}"/"]{path}["?"{query}]["#"fragment]
```

Eine URI beschreibt stets ein Schema (`scheme`) und einen Pfad (`path`), der jedoch leer sein kann. Die Angabe der Bestandteile `authority`, `query` und `fragment` ist optional. Zur weiteren Erläuterung seien Beispiele gültiger URIs dargestellt (in Anlehnung an [BFM05, S. 7]):

```
http://www.e-car.org/vertrieb
ftp://ftp.is.co.za/docs/rfc?number=3986#genericsyntax
mailto:M.Wolf@seda.example.de
tel:+49-928-187-325
urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

Das *Schema* definiert den URI-Kontext und spezifiziert die weiterfolgende Syntax des Identifikators [BFM05, S. 17]. Als Schemata können beispielsweise Protokolle wie `http`, oder Notationsansätze wie `URN`, angegeben werden. Ein *Pfad* beschreibt die eindeutige Identifikation einer Ressource in einem hierarchisch organisierten System und trennt die einzelnen Hierarchiestufen durch "/" (z. B. `/docs/rfc`).

Viele Schemata spezifizieren zudem einen `authority`-Teil zur Identifikation einer Instanz, auf deren Basis eine Verwaltung der Namensräume innerhalb des Schemas erfolgt. Bekanntes Beispiel hierfür ist das Domain Name System (DNS) ([Mock87a], [Mock87b]), das einen Verzeichnisdienst zur Registrierung von eindeutigen Namen (Domains) und deren Identifikation in (IP-basierten) Netzwerken implementiert. In einer Abfrage (`query`) können Kriterien zur Selektion einer Ressourcenmenge definiert werden, die durch die Angabe des Pfades nicht

[9] Webseite der Internet Engineering Task Force (IETF): http://www.ietf.org/ (Abruf am 25. März 2015).

eindeutig identifizierbar sind [BFM05, S. 23]. Ein Fragment (`fragment`) bestimmt eine Untermenge oder referenziert eine Stelle innerhalb der Ressource [BFM05, S. 24 f.].

Den Zusammenhang zwischen URI, URL und URN visualisiert Abbildung 3.8. Er wird im weiteren Verlauf erläutert [ECP+13, S. 69 f.]:

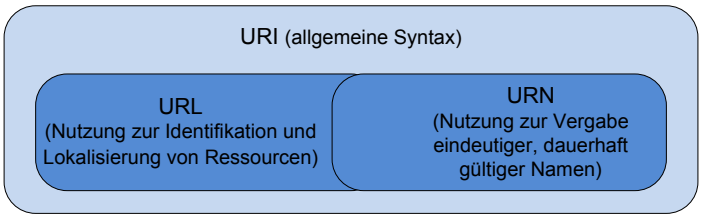


Abbildung 3.8: Zusammenhang zwischen URI, URL und URN
(in Anlehnung an [Wiki14], [ECP+13, S. 70], eigene Darstellung)

- Eine URI beschreibt eine Zeichenfolge zur Erstellung eindeutiger Identifikatoren, die nach einer definierten Syntax strukturiert ist.
- URLs sind URIs, die für die eindeutige Identifikation und Lokalisierung von Ressourcen in einem Netzwerk geeignet sind. Eine URL spezifiziert folglich alle notwendigen Informationen, die für einen Zugriff auf die referenzierte Ressource und den Aufruf von Methoden darunterliegender Dienste benötigt werden. Aus diesem Grund sind die URIs zur eindeutigen Identifikation von Ressourcen in einer REST-Architektur im Allgemeinen auch URLs.
- URNs sind URIs zur eindeutigen und dauerhaft gültigen Vergabe von Namen an eine Ressource. Folglich muss eine URN keine Informationen enthalten, um Ressourcen in einem Netzwerk lokalisierbar und referenzierbar zu machen.

Der *International Resource Identifier* (IRI) Standard erweitert den ASCII-Zeichensatz der URI um eine Vielzahl an Elementen aus der Unicode-Zeichenmenge [DuSu05].

Die globale Adressierbarkeit von Ressourcen ist eine Voraussetzung zur Realisierung von Zugriffen auf die Methoden der Ressourcenschnittstelle und damit die Nutzung der zugrundeliegenden Dienste. Im Umfeld von REST kommt HTTP als Anwendungsprotokoll zum Einsatz (z. B. [Tilk11], [WPR10], [Fiel00, S. 116 ff.]). Aus diesem Grund gehen die spezifizierten URIs meist auch mit den Anforderungen des URL-Schemas konform. Beide Begriffe werden im weiteren Verlauf deshalb gleichbedeutend verwendet.

3.4.2 Hypertext Transfer Protocol

Das *Hypertext Transfer Protocol* (HTTP) ist das zentrale Kommunikationsprotokoll für die Übertragung von Informationen im WWW [Fink13b]. HTTP setzt dabei auf den grundlegenden Kommunikationsprotokollen des Internets auf und implementiert eine Schicht, deren Schnittstelle von Programmen zum Austausch von anwendungsspezifischen Daten genutzt werden kann. Nach dem ISO/OSI-Referenzmodell (z. B. [FeSi13, S. 416 ff.]) lässt sich HTTP als *Anwendungsprotokoll* zur Durchführung von Aufgaben anwendungsorientierter Schichten

klassifizieren [Fink13b]. Aktuell liegt HTTP in der Version 1.1 vor und wird als Standard der IETF[10] verwaltet [FGM+99].

HTTP ist ein *zustandsloses Protokoll*, das die Kommunikation zwischen Client und Server nach dem *Request-/Response-Prinzip* durchführt. Für den Zugriff auf Ressourcen stellt der Client eine Anfrage (Request) an den Server. Dieser verarbeitet die Anfrage und schickt eine Antwort (Response) zurück an den Client. Anfragen und Antworten werden in Form von HTTP-Nachrichten übermittelt. Nach dem Request-/Response-Schema wird der Zustand einer Verbindung nur für die Dauer einer Anfrageabwicklung aufrechterhalten [Fink13b].

Eine *HTTP-Nachricht* besteht aus einem *Nachrichtenkopf* (message header, header) und einem *Nachrichtenkörper* (message body, body). Während der Nachrichtenkörper die eigentlichen Nutzdaten, z. B. die HTML-Datei einer angeforderten Webseite, enthält, spezifiziert der Nachrichtenkopf die Metadaten der Nachricht. Typische Informationen des Headers sind die Adresse der aufzurufenden Ressource im Netzwerk (HTTP-URI), die verwendete HTTP-Methode oder das Inhaltsformat (Content-Type) des Nachrichtenkörpers [FGM+99, S. 17 ff.]. Die Struktur der zwischen Client (Web-Browser) und (Web-)Server ausgetauschten http-Nachrichten stellen Quellcode 3.1 und Quellcode 3.2 am Beispiel eines Webseitenaufrufs dar.

```
GET /seda  HTTP/1.1
Host: www.uni-bamberg.de
Accept: text/html,application/xhtml+xml
...
```

Quellcode 3.1: Beispiel eines HTTP-Requests

```
HTTP Version: HTTP/1.1
Status Code: 200 OK
Content-Encoding: gzip
Content-Length: 6711
Content-Type: text/html; charset=utf-8
Date: Fri, 24 Feb 2015 07:43:46 GMT
Server: Apache/2.2.22 (Ubuntu)

<html> ... </html>
```

Quellcode 3.2: Beispiel eines HTTP-Response

Am Beispiel der HTTP-Anfrage ist ersichtlich, dass das Anwendungsprotokoll URIs zur Adressierung von Ressourcen im WWW verwendet (Zeile 1 und 2 in Quellcode 3.1). Daneben werden die Art der HTTP-Methode (GET) sowie die akzeptierten Repräsentationsformate (*Accept*) der angefragten Ressource (z. B. Webseite in HTML-Format) spezifiziert. Die HTTP-Antwort beschreibt im Nachrichtenkopf den Status der Anfrageverarbeitung in Form eines Response Codes (hier `200 OK`) sowie weitere Eigenschaften der Nachricht, z. B. das Format des übermittelten Nachrichtenkörpers. Der Körper selbst enthält den HTML-Code der aufgerufenen Webseite (`<html> … </html>`). Mit dem Empfang der Antwortnachricht wird die Verbindung zwischen Client und Server beendet.

[10] W3C HTTP - Hypertext Transfer Protocol – Overview. Webseite: http://www.w3.org/Protocols/ (Abruf am 25. März 2015).

Zur Realisierung des Zugriffs auf Ressourcen definiert HTTP eine generische Menge von insgesamt acht Anfrage-Methoden. Tabelle 3.1 listet sechs *HTTP-Methoden* auf, die in Zusammenhang mit der Entwicklung von RESTful SOA zum Einsatz kommen [FGM+99, S. 51 ff.].

Methode	Erläuterung und Eigenschaften
GET	Abfrage der Repräsentation (Informationen) einer Ressource, die durch die URI des Requests identifiziert wird. Die Antwort an den Server enthält, im Falle einer erfolgreichen Verarbeitung, die angefragte Repräsentation im Nachrichtenkörper.
	Sicher: Ja; *Idempotent*: Ja; *Sichtbare Semantik*: Ja
POST	Fragt i. d. R. die Erstellung einer neuen Ressource unterhalb des Pfades an, den die URI identifiziert. Die im Nachrichtenkörper übermittelten Daten des POST-Requests werden dann vom Server als neue Ressource persistiert.
	Sicher: Nein; *Idempotent*: Nein; *Sichtbare Semantik*: Nein
PUT	Fragt die Aktualisierung einer, durch die URI des Requests identifizierten, Ressource auf Basis der übermittelten Daten des PUT-Nachrichtkörpers an. Adressiert die URI keine existierende Ressource, so wird eine neue Instanz angelegt.
	Sicher: Nein; *Idempotent*: Ja; *Sichtbare Semantik*: Ja
DELETE	Fragt die Löschung der Ressource an, welche die URI des Requests identifiziert.
	Sicher: Nein; *Idempotent*: Ja; *Sichtbare Semantik*: Ja
OPTIONS	Abfrage von Metadaten der Optionen, die bezüglich einer Interaktion des Clients mit der identifizierten Ressource des Servers bestehen (z. B. unterstützte Methoden).
	Sicher: Ja; *Idempotent*: Ja; *Sichtbare Semantik*: Ja
HEAD	Entspricht dem GET-Request mit dem Unterschied, dass kein Nachrichtenkörper in der HTTP-Antwort enthalten ist. Die übersandten Metadaten des Kopfes der Antwortnachricht stimmen mit den Informationen einer korrespondierenden GET-Anfrage überein.
	Sicher: Ja; *Idempotent*: Ja; *Sichtbare Semantik*: Ja

Tabelle 3.1: Zusammenfassung der HTTP-Methoden

HTTP-Methoden weisen eine Reihe von Eigenschaften auf [Tilk11, S. 56]. Eine Methode ist *sicher*, wenn mit ihrer Ausführung keine Verpflichtungen, i.S.v. Auswirkungen auf den Zustand des Systems, eingegangen werden. Erzeugt das mehrmalige Aufrufen einer Methode (unter gleichen Voraussetzungen) stets die gleichen Ergebnisse, so ist sie *idempotent*. Abgesehen von POST besitzen die genannten Methoden eine s*ichtbare Semantik*. Ein Aufrufer hat somit Kenntnis über ihre Wirkung. Darüber hinaus sind GET und HEAD *Cache-fähig* (z. B. [Burk13, S. 7 ff.], [Davi12], [Tilk11, S. 56], [Alla10, S. 10]).

Eine zentrale Bedeutung für die Implementierung der einheitlichen Komponentenschnittstelle in REST-Architekturen besitzen die HTTP-Methoden *GET, POST, PUT* und *DELETE*. Ihre Funktionalität korrespondiert mit den klassischen CRUD-Operationen (Create-Read-Update-Delete*)* der Datenverwaltung. Zusammen mit der Ressourcenorientierung sind sie das grundlegende Konzept zur Realisierung der anwendungsspezifischen Funktionalität in einem verteilten Anwendungssystem (z. B. [WPR10, S. 57 f.], [PZL08, S. 807], [RiRu07, S. 97 f.]).

Bei der Abfrage der Metadaten von Ressourcen können zudem die Operationen HEAD und OPTIONS in REST-Anwendungen eingesetzt werden. Durch die Bereitstellung von Metadaten spielen sie eine unterstützende Rolle, um z. B. die Entdeckung von Services oder deren

Nutzung vorzubereiten (z. B. [WPR14, S. 18 ff.], [VSD+12], [StAl11]). Bei der Realisierung von RESTful Services stehen jedoch die CRUD-Operationen im Mittelpunkt der Entwicklung.
Die HTTP-Spezifikation definiert mit TRACE und CONNECT, neben den bereits genannten, noch zwei weitere Methoden. TRACE unterstützt die Nachverfolgbarkeit von Nachrichten über mehrere Teilnehmer hinweg. Die Erstellung von Verbindungen über eine Proxy erfolgt mit CONNECT. Für die Entwicklung von REST-Architekturen besitzen diese jedoch nur eine untergeordnete Rolle ([Burk13, S. 8], [Tilk11, S. 56], [WPR10], [Alla10]). Zur tiefergehenden Erläuterung der HTTP-Methoden und ihrer Eigenschaften sei auf den aktuellen Standard [FGM+99] und die erläuternde Literatur (z. B. [GoTo02]) verwiesen.

Das Anwendungsprotokoll HTTP bildet aufgrund seiner Merkmale der zustandslosen Client-Server-Kommunikation und der definierten, generischen Methodenmenge die technologische Basis für die Realisierung von REST-Architekturen (z. B. [ECP+13], [Tilk11], [WPR10]).

3.4.3 Repräsentationsformate

Eine Ressource interagiert mit ihrer Umwelt über die Bereitstellung einer oder mehrerer Repräsentationen, durch die auch eine Manipulation des internen Zustands erfolgen kann. *Repräsentationen* sind somit die öffentliche Darstellung von Ressourcenzuständen in einem definierten *Format* ([WPR10, S. 4 f.], [Fiel00, S. 90 f.]).

Im Umfeld des WWW und damit auch REST-Anwendungen werden *Repräsentationsformate* auf Basis von Internet Media Types[11] der IANA[12] genutzt [Fiel00, S. 91]. Die Standards XML und JSON sind im Umfeld von Web-Anwendungen die am weitest verbreiteten Repräsentationsformate für eine Computer-Computer-Kommunikation (CCK) [MPD10]. Die Beschreibung von Informationen zur Mensch-Computer-Kommunikation (MCK) erfolgt mit dem Standard HTML.
Darüber hinaus haben sich eine Reihe alternativer Repräsentationsformate wie Atom und AtomPub ([NoSa05], [GrHo07]), XHTML [BGM+10] oder auch JSON-LD[13] [SKL14] im Umfeld von REST-Architekturen etabliert. Für eine tiefergehende Erläuterung dieser Formate sei auf die einschlägige Literatur verwiesen (z. B. [Tilk11], [Alla10], [WPR10], [LaGü13], [LaGü12]).

Da in REST-Architekturen häufig die allgemeinen Standards XML, JSON und HTML als Repräsentationsformate zum Einsatz kommen, werden diese in den nachfolgenden Abschnitten eingeführt. Sie werden auch in allen Beispielen der vorliegenden Arbeit als Formate verwendet.

XML

Die XML (*eXtensible Markup Language*) [BPS+08] ist eine Metasprache zur Definition von Auszeichnungssprachen (Markup-Sprachen) für Dokumente beliebiger Anwendungsdomänen

[11] *Internet Media Types* (auch MIME-Types) [IANA14] klassifizieren den Inhalt des Körpers einer Nachricht im Internet. Vom Sender wird hierzu dem Nachrichtenempfänger der Typ der übermittelten Daten im Feld Content-Type der Nachricht mitgeteilt (vgl. Quellcode 3.2).

[12] Die *Internet Assigned Numbers Authority* (IANA) ist für die globale Verwaltung von IP-Adressen, DNS-Roots und anderen Internet-Ressourcen verantwortlich (Webseite: http://www.iana.org/ (Abruf am 25. März 2015)).

[13] Eine Einführung in JSON-LD findet sich unter http://json-ld.org/ (Abruf am 25. März 2015).

[Neum13]. Das ursprüngliche Ziel der Entwicklung von XML war die Bereitstellung eines einfachen, flexiblen und erweiterbaren Standards für den Austausch von strukturierten Dokumenten im Internet. Auf Grundlage der XML wird auch die Grammatik der Sprache, mittels Dokumenttypdefinitionen (DTD, Document Type Definition) oder XML-Schemata, spezifiziert, welche strukturelle und inhaltliche Anforderungen zur Erstellung von XML-Dokumenten vorgeben. XML liefert eine allgemeine Syntax zur hierarchischen Strukturierung von Inhalten und Ableitung dieser Sprachen ([Neum13], [Poma12, S. 127], [HKR+08, S. 17 ff.]). XML ist eine Untermenge der Standard Generalized Markup Language (SGML) und wird seit dem Jahr 1998 von der W3C als Empfehlung (Recommendation) entwickelt. Aktuell liegt die XML-Empfehlung in der fünften Version vor [BPS+08][14].

Aufgrund ihrer Flexibilität besitzt die XML ein weites Einsatzfeld und ist die Grundlage einer Vielzahl bedeutender Standards. So bildet die XML z. B. die Basissprache für die Standards SOAP, WSDL und BPEL[15] zur Spezifikation von Web-Services oder die Sprache RDF[16] des Semantic Webs ([Neum13], [Tilk11, S. 84 ff.]).

XML-Dokumente bestehen aus einem (optionalen) Prolog und dem Dokumenteninhalt. Im Prolog können grundlegende Dokumenteneigenschaften, z. B. XML-Version oder genutzte Zeichencodierung, in der XML-Deklaration sowie die verwendete DTD spezifiziert werden. Den Inhalt eines XML-Dokuments beschreibt ein hierarchisch strukturierter Baum mit genau einem Wurzelelement. Jeder Knoten des XML-Baumes ist ein XML-Element, das durch einen *Tag* geöffnet (z. B. `<Person>`) und geschlossen (z. B. `</Person>`) wird. Optional besitzt jedes Element ein oder mehrere XML-Attribute (z. B. `Anrede=""`). Zwischen den Tags steht der Inhalt des Elements ([Poma12, S. 129 ff.], [MRS09, S. 197 ff.]). Quellcode 3.3 visualisiert den Aufbau eines XML-Dokuments am Beispiel einer Person.

```
<?xml version="1.0" encoding="utf-8"?>

<Personen>
      <Person Anrede="Herr">
            <Name>Max Mustermann</Name>
            <Wohnort>Musterstadt</Wohnort>
      </Person>
      <Person>...</Person>
</Personen>
```

Quellcode 3.3: Beispiel eines XML-Dokuments

Die Formulierung *syntaktischer Anforderungen* an die Struktur einer Dokumentinstanz erfolgt meist mit Hilfe von DTDs oder XML-Schemata. Eine DTD definiert die erlaubte Struktur eines XML-Dokuments in Form einer kontextfreien Grammatik, die beispielsweise die Verschachtelung der Elemente oder die verfügbaren Element-Attribute festlegt. Eine mächtigere, aber

[14] Weiterführende Informationen zu XML finden sich auf Webseite der W3C unter http://www.w3.org/XML/ (Abruf am 25. März 2015)

[15] Zu den Technologien von Web-Services sei auf die weiterführende Literatur, z. B. in MELZER ET AL [Melz10] oder PAPAZOGLOU [Papa08], verwiesen.

[16] Auch das Resource Description Framework (RDF) wird von der W3C verwaltet. Informationen finden sich z. B. auf der RDF-Webseite unter http://www.w3.org/RDF (Abruf am 25. März 2015).

auch komplexere Alternative zu DTD sind XML-Schemata, die in weitaus detaillierterer Form die Struktur des XML-Dokuments spezifizieren können. XML-Schemata sind wiederum selbst XML-Dokumente ([HKR+08, S. 22 ff.], [Poma12, S. 132 ff.]).

Zur Vermeidung von Namenskonflikten, meist sind dies Mehrdeutigkeiten zwischen Auszeichnungselementen in einem XML-Dokument, wurde das Konzept der Namensräume (engl. Namespaces) als W3C-Empfehlung erarbeitet [BHL+09]. Durch die Verwendung von Namensräumen können XML-Namen von Elementen und Attributen in Mengen gegliedert werden. Diese definieren sogenannte XML-Vokabulare und sind durch eine URI identifizierbar. Verschiedene Vokabulare können darauf aufbauen und in XML-Dokumenten wiederverwendet oder auch kombiniert werden (z. B. [HKR+08, S. 25 ff.]).

XML-Dokumente sind auf die korrekte Verwendung der Auszeichnungen ihrer XML-Elemente überprüfbar. Ein korrekt ausgezeichnetes Dokument wird als *wohlgeformt* bezeichnet. Werden die Elemente eines Dokuments konform in Bezug auf seine Dokumenttypdefinition oder Schema deklariert, ist dieses *gültig*. Eine Prüfung auf Wohlgeformtheit und Gültigkeit erfolgt meist automatisiert durch den Einsatz von XML-Parsern ([HKR+08, S. 22], [Poma12, S. 127 ff.]).

JSON

JSON (*JavaScript Object Notation*) ist ein textbasiertes, sprachunabhängiges Dokumentformat zum Austausch von strukturierten Daten zwischen Anwendungsprogrammen. Die JSON-Spezifikation wurde aus der Programmiersprache JavaScript[17] abgeleitet und soll stets als gültiges JavaScript interpretierbar sein. Aktuell wird JSON in den zwei konkurrierenden Standards *RFC-4627* der IETF [Croc06] und *ECMA-404* [ECMA13] spezifiziert.

Die Zielsetzung von JSON ist es, durch eine kleine und möglichst einfache Menge an Strukturierungsregeln die Repräsentation von Daten in einer strukturierten Form bereitzustellen. Hierfür definiert die Notation vier primitive Typen (string, number, boolean und null) sowie zwei strukturierte Typen (object, array). Als Syntaxzeichen werden geschweifte Klammern ("`{}`"), eckige Klammern ("`[]`"), Kommas ("`,`") und Doppelpunkt ("`:`") verwendet ([Croc06, S. 2], [ECMA13, S. 1]). Aufgrund seiner einfachen Syntax wird JSON als einfache, weniger universelle, XML-Alternative zur Serialisierung und zum Austausch strukturierter Daten angesehen ([Poma12, S. 150 f.], [RiRu07, S. 44 f.]). Die Notation besitzt besonders im Umfeld von webbasierten Anwendungen eine hohe Verbreitung und wird von vielen Programmiersprachen unterstützt[18].

Ein **JSON-Dokument** besteht aus einer Menge von Einträgen (Elementen). Jedes Element ist entweder ein Objekt oder ein Array. Objekte beschreiben eine ungeordnete Menge an Name/Wert-Paaren. Namen sind immer vom Datentyp `String`. Werte sind von einem beliebigen (primitiven oder strukturierten) Typ. Ein Array ist eine geordnete Liste von Werten. Als Beispiel eines JSON-Dokuments stellt der Quellcode 3.4 die Datenstruktur einer Buchung

[17] Basierend auf dem Standard ECMA-262 – 3.Edition – Dezember 1999 (nach [ECMA11])

[18] Eine Einführung in das JSON-Dokumentformat und Auflistung unterstützender Programmiersprachen bzw. Bibliotheken findet sich unter http://json.org (Abruf am 25. März 2015).

als JSON-Text dar. Das Buchungsobjekt besteht hier aus drei Name/Wert-Paaren. Der Wert des dritten Name/Wert-Paares ist ein Array (`"references" : []`), welches wiederum zwei Objekte enthält, die URI-Links auf die Ressourcen Buchung (`"self"`) und Kunde (`"kunde"`) beschreiben.

```
{
  "buchungID" : 1,
  "datum" : "30.06.2014",
  "references" : [
      {"type":"self","uri":"/bookings/1"},
      {"type":"kunde", "uri":"/customers/815"} ]
}
```

Quellcode 3.4: Beispiel eines JSON-Dokuments

In der IETF-Spezifikation wird der Aufbau eines JSON-Textes mit einer Menge von Regeln definiert [Croc06, S. 2 ff.]. Die Struktur eines JSON-Dokuments veranschaulicht Abbildung 3.9 in Form eines Metamodells (eigene Darstellung).

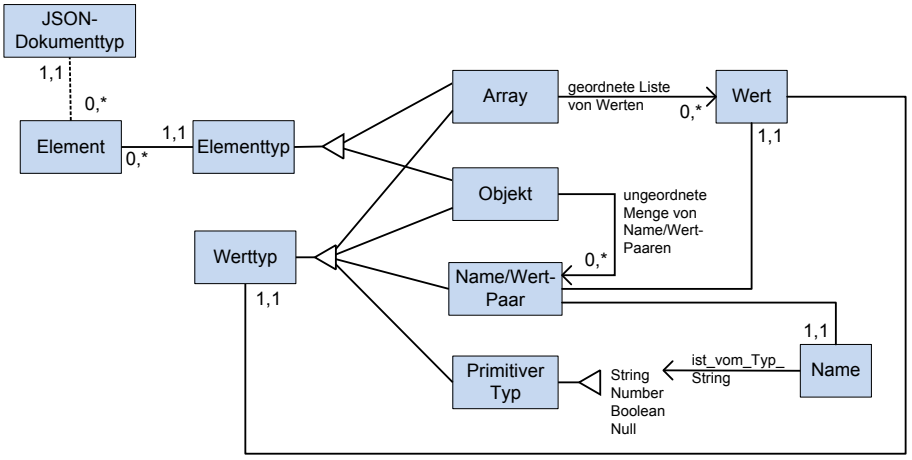


Abbildung 3.9: Metamodell des JSON-Dokumenttyps

In REST-Architekturen wird JSON häufig zur Repräsentation der Datenstrukturen von Ressourcen eingesetzt. Wichtige Gründe hierfür sind die einfache Syntax und die gute Unterstützung der maschinellen Verarbeitung durch eine Vielzahl von Web-Programmiersprachen[18]. Des Weiteren wird JSON als Serialisierungsformat zur direkten Persistierung von Dokumenten in *dokumentorientierten Datenbanken* verwendet. Diese stellen zunehmend eine bedeutende Alternative zu den klassischen relationalen Datenbanken im Umfeld von webbasierten AwS dar (z. B. [NAK14], [LaCr13], [Fris12b], [MCB+11]). Im weiteren Verlauf der Arbeit wird JSON als Standardformat zur Repräsentation von Ressourcenzuständen für die maschinelle Verarbeitung verwendet.

HTML

Die HTML (*Hypertext Markup Language*) ist eine Auszeichnungssprache zur Strukturierung von Dokumenten in textueller Form. Im WWW ist HTML die Standard-Sprache zur Gestaltung des Aufbaus und Inhalts von Webseiten. HTML wird von der W3C entwickelt und liegt seit 1999 in der Version 4.01 [RLJ99] vor. Die Weiterentwicklung zur Version HTML 5 befindet sich aktuell im Entwurfsstadium [BFL+14].

Ein **HTML-Dokument** ist in die Bereiche *Dokumenttyp-Deklaration*, *Dokumentkopf* (<HEAD>) sowie *Dokumentkörper* (<BODY>) untergliedert. Kopf und Körper eines HTML-Dokuments werden zudem von einem HTML-Root-Tag (<HTML>) umschlossen. Die Dokumenttyp-Deklaration beschreibt die verwendete HTML-Version und definiert durch Angabe der DTD (strict.dtd) die erlaubten Elemente und Attribute des Dokuments. Der Kopf enthält Information über das Dokument wie z. B. Titel oder Metadaten (Autor, Schlagwörter, etc.). Der eigentliche Inhalt eines HTML-Dokuments wird in seinem Körper beschrieben und durch HTML-Tags strukturiert und gestaltet. Die konkrete Darstellung des Inhalts ist dabei nicht festgelegt und wird erst durch die Interpretation mit Hilfe eines geeigneten Werkzeuges (Browser) bestimmt [RLJ99, Abschnitte 7.4 und 7.5].

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">
<HTML>
    <HEAD>
        <TITLE>Beispiel HTML-Dokument</TITLE>
        <META name="author" content="Max Muster">
    </HEAD>
    <BODY>
        <P>Hallo Welt!</P>
        <P><A HREF="http://www.uni-bamberg.de">Hyperlink</A></P>
    </BODY>
</HTML>
```

Quellcode 3.5: Beispiel eines HTML-Dokuments (in Anlehnung an [RLJ99, Abschnitt 7.1])

Durch die Spezifikation von *Hyperlinks* (<A>-Tag) werden in HTML Referenzen auf adressierbare Informationseinheiten definiert. Dies ermöglicht den Aufbau eines *Hypermedia*-Systems, das aus einer Menge von Knoten (Informationseinheiten, Ressourcen) und gerichteten Kanten (Links) besteht. Den Zielknoten eines Links adressiert seine URI ([Fink13a], [RLJ99, Abschnitt 12]).

HTML ist im WWW-Umfeld der Standard zur Gestaltung und Strukturierung von Informationen in menschenlesbarer Form. Zudem ermöglicht das Konzept des Hyperlinks die einfache Steuerung des Ausführungszustandes einer Anwendung durch den (personellen) Nutzer. Im weiteren Verlauf der Arbeit wird HTML deshalb als Standardformat zur Repräsentation von Ressourcen für die Mensch-Computer-Kommunikation verwendet.

3.5 Zusammenfassung

Das dritte Kapitel führt die RESTful SOA als Zielarchitektur der modellbasierten Entwicklung in dieser Arbeit ein. Beginnend mit einer Klärung des Begriffs der service-orientierten Architektur erfolgen die Erarbeitung grundlegender Merkmale sowie die Bestimmung allgemeiner Beschreibungsebenen für die Analyse, Gestaltung und Pflege von SOA. Das SOA-Architekturkonzept ist unabhängig vom Einsatz konkreter Technologien und Standards. Aus fachlicher Sicht dienen SOA der Unterstützung von Geschäftsprozessen betrieblicher Systeme und stellen ihre Funktionalität hierzu in Form von Diensten bereit. Diese sind nach der Art der Funktionalität grundsätzlich als Vorgangs- und Entitätsdienste klassifizierbar.

Die konkrete Gestaltung von SOA geht in dieser Arbeit konform mit dem Architekturstil REST, der insgesamt fünf Bedingungen an die Realisierung verteilter Hypermedia-Systeme stellt. Im Zusammenhang mit der Entwicklung von SOA und ihrer Komponenten besitzt insbesondere die Bedingung der einheitlichen Schnittstelle große Bedeutung. Die Anwendung der einheitlichen Schnittstelle führt zu lose gekoppelten Ressourcen mit Standard-Operationen und eindeutigem Identifikator als zentrale Systemkomponenten des AwS. Eine Umsetzung der REST-Bedingungen in webbasierten Systemen fußt im Kern auf dem Einsatz der Standards HTTP, URI und den Repräsentationsformaten XML, JSON und HTML.

Durch die Anwendung des REST-Stils auf das SOA-Architekturkonzept wird die RESTful SOA gebildet. Die Schnittstelle von RESTful Services ist in Form von Ressourcen realisiert, auf die im Rahmen der Service-Nutzung zugegriffen wird. Die fachliche Funktionalität eines unterstützten Geschäftsprozesses stellen RESTful SOA nicht durch eine Menge anwendungsspezifischer Operationen, sondern in Form anwendungsspezifischer Ressourcen mit Standard-Operationen bereit. Der Aufbau von RESTful SOA wird unter dem Blickwinkel der Außen- und Innenperspektive untersucht und ihr grundlegender konzeptueller sowie softwaretechnischer Aufbau eingeführt.

Der Aufbau, die Bestandteile und die zentralen Eigenschaften von RESTful SOA, die für eine modellbasierte Spezifikation erforderlich sind, werden abschließend noch einmal genauer beleuchtet. Eine Annahme dieser Arbeit ist, dass RESTful Services bzw. ihre Ressourcen geeignet sind, eine modellbasierte Überführung von SOM-Geschäftsprozessbausteinen in korrespondierende Komponenten der Softwareebene und die flexible Anpassbarkeit der entwickelten Systemarchitektur zu unterstützen. Die vorausgegangenen Untersuchungen der Eigenschaften von RESTful SOA stützen diese Annahme. Die Veröffentlichung einer einheitlichen Schnittstelle legt den Schwerpunkt der Systemgestaltung auf die inhaltliche Bedeutung von Ressourcen bzw. der RESTful Services. Die Spezifikation von Ressourcen ist maßgeblich durch ihren fachlichen Ursprung auf der Ebene des betrieblichen Objektsystems bestimmt. Die Nutzung der implementierten Services von RESTful SOA setzt die Kenntnis der Semantik der unterstützten Bestandteile im Geschäftsprozessmodell voraus. Der Austausch von Ressourcen-Repräsentationen in Form von standardisierten HTTP-Nachrichten führt zu einer losen Kopplung der Systemkomponenten und der Dokumentenorientierung ihrer Interaktion. Die Ressourcen- und Dokumentenorientierung sind die zentralen Konzepte der RESTful SOA, um die Flexibilität und zielgerichtete Anpassbarkeit eines entwickelten betrieblichen Anwendungssystems zu fördern. Die vorangegangenen Ausführungen bilden den Ausgangspunkt für die konzeptuelle Modellierung der RESTful SOA und die Gestaltung des Architekturmodells der Entwicklungsmethodik in Kapitel 5.

4 Flexible SOM-Geschäftsprozessmodelle

Flexible Geschäftsprozessmodelle auf Basis des Semantischen Objektmodells (SOM) bilden in dieser Arbeit den Ausgangspunkt der modellbasierten Spezifikation von RESTful SOA. Im vierten Kapitel wird zunächst die SOM-Methodik als methodischer Rahmen zur Modellierung von Geschäftsprozessen und modellbasierten Spezifikation der Anwendungssystemebene eingeführt. Anschließend wird die Flexibilität in betrieblichen Systemen untersucht und, auf diesem Verständnis aufbauend, ein Flexibilitätsbegriff für SOM-Geschäftsprozessmodelle bestimmt. Zuletzt wird die Fallstudie „Flugbuchung" vorgestellt, die zur Veranschaulichung der weiteren konzeptionellen Ausführungen dient.

4.1 Modellierung betrieblicher Informationssysteme in der SOM-Methodik

Das *Semantische Objektmodell* (SOM) ([FeSi90], [FeSi91], [FeSi95], [FeSi06]) ist ein objekt- und geschäftsprozessorientierter Ansatz für die Modellierung betrieblicher Systeme [FeSi13, S. 194]. SOM spezifiziert den methodischen Rahmen, auf dessen Basis die modellbasierte Überführung von Geschäftsprozessmodellen in eine objektorientierte AwS-Spezifikation erfolgt. In der vorliegenden Arbeit bildet SOM die methodische Basis der Systementwicklung.

4.1.1 Die SOM-Methodik

Das Semantische Objektmodell ist ein umfassender Modellierungsansatz für betriebliche Systeme [FeSi95]. Zur Unterstützung der Strukturierung betrieblicher Systeme und der Bewältigung ihrer hohen Komplexität bei der Durchführung der Modellierungsaufgabe wird der SOM-Ansatz um eine Unternehmensarchitektur und ein Vorgehensmodell ergänzt. Modellierungsansatz, Unternehmensarchitektur und Vorgehensmodell werden zusammen als *SOM-Methodik* (Methodik des Semantischen Objektmodells) bezeichnet [FeSi13, S. 194 f.].

Die Modellierungsreichweite von SOM umfasst das gesamte betriebliche System. Sie wird in der vorliegenden Arbeit jedoch auf das informationsverarbeitende Teilsystem (IS) eingeschränkt, da dessen automatisierbaren Aufgaben den Ausgangpunkt für die schrittweise Entwicklung der AwS beschreiben. Das Basissystem, und damit das Aufgabenobjekt „Nicht-Information", ist nicht Gegenstand weiterer Untersuchungen (Abschnitt 2.1.2).

Die Modellierung in der SOM-Methodik richtet sich zur Unterstützung der Erfassung des Objektsystems und der Spezifikation des Modellsystems betrieblicher Systeme an folgender übergeordneter *Metapher* aus:

> „*Ein betriebliches System wird aus Außensicht als offenes, zielgerichtetes und soziotechnisches System betrachtet. Aus Innensicht liegt ein verteiltes System vor, bestehend aus einer Menge autonomer, lose gekoppelter Objekte, die im Hinblick auf die Erreichung übergeordneter Ziele kooperieren*" [FeSi13, S. 197].

Die übergeordnete Metapher bildet die Grundlage für die Strukturierung betrieblicher Systeme in der SOM-Unternehmensarchitektur und des Vorgehens zur Durchführung der Modellierung (Vorgehensmodell). Die Modellbildung auf den Ebenen der Unternehmensarchitektur wird in der SOM-Methodik anhand des SOM-Vorgehensmodells durchgeführt [FeSi13, S. 197 f.].

SOM-Unternehmensarchitektur

Das Modellsystem betrieblicher Systeme wird in der SOM-Methodik in drei aufeinander aufbauende Teilmodellsysteme untergliedert. Diese beschreiben das betrachtete Objektsystem vollständig aus einer bestimmten Perspektive. Die Abgrenzung der Teilmodellsysteme erfolgt dabei anhand der Kriterien *Außen-* und *Innenperspektive* sowie *Aufgaben-* und *Aufgabenträgerebene* [FeSi13, S. 195 f.]. Die einzelnen Teilmodellsysteme werden jeweils auf einer Modellebene beschrieben. Die drei Modellebenen und die Beziehungen zwischen den Ebenen erfasst die (*SOM-) Unternehmensarchitektur* in Form eines integrierten Architekturmodells ([FeSi95], [FeSi06]). Die Unternehmensarchitektur stellt somit eine Extension des generischen Architekturmodells dar ([Sinz95] sowie Abschnitt 2.1.6).

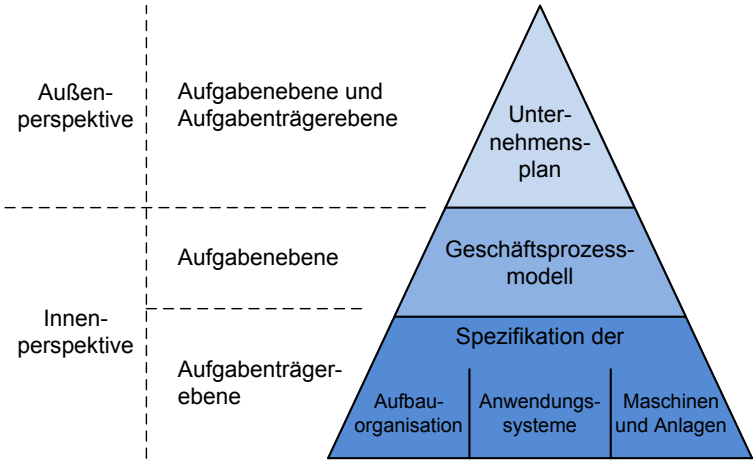


Abbildung 4.1: SOM-Unternehmensarchitektur [FeSi13, S. 195]

Abbildung 4.1 zeigt den grundlegenden Aufbau der *Unternehmensarchitektur*, welche die Modellebenen *Unternehmensplan*, *Geschäftsprozessmodell* und *Ressourcenmodell* umfasst. Die drei Modellebenen werden im Folgenden genauer vorgestellt:

1. **Unternehmensplan**: Der Unternehmensplan beschreibt die Aufgaben- und Aufgabenträgerebene des betrieblichen Systems aus der Außenperspektive. Hierzu werden im Unternehmensplan *Diskurswelt* und *Umwelt* voneinander abgegrenzt und ihre *Leistungsbeziehungen* definiert. Die grundlegende Sichtweise für die Erstellung des Unternehmensplans legt die Metapher der globalen Unternehmensaufgabe fest. Die Unternehmensaufgabe wird anhand der Merkmale Aufgabenobjekt (Diskurswelt), Aufgabenziele (Sach- und Formalziele), Leistungsbeziehungen zur Umwelt sowie benötigter Ressourcen zur Aufgabendurchführung beschrieben. Zudem werden Strategien und relevante Rahmenbedingungen erfasst [FeSi13, S. 196].

2. **Geschäftsprozessmodell**: Das Geschäftsprozessmodell beschreibt die Aufgabenebene des betrieblichen Systems aus der Innenperspektive. Zielsetzung der Modellebene ist die Modellierung des *Lösungsverfahrens* zur Durchführung der Unternehmensaufgabe, die im

Unternehmensplan definiert ist. Das Lösungsverfahren der Unternehmensaufgabe wird dabei als *Geschäftsprozessmodell* in Form einer Menge von Prozessen beschrieben, die entweder als Hauptprozesse über Leistungsbeziehungen mit der Umwelt interagieren und damit zur Sachzielerfüllung beitragen, oder als Serviceprozesse selbst Leistungen für Haupt- oder Serviceprozesse zur Verfügung stellen. Der Modellierung von SOM-GPM liegt die Metapher eines verteilten Systems zugrunde, das aus einer Menge autonomer und lose gekoppelter Komponenten besteht, die sich koordinieren, um gemeinsame Ziele zu erfüllen [FeSi13, S. 196 f.].

3. **Ressourcenmodell**: Das Ressourcenmodell beschreibt die Aufgabenträgerebene des betrieblichen Systems aus der Innenperspektive. Die Modellierung ist an der allgemeinen Metapher des sozio-technischen Systems ausgerichtet, in dem die Durchführung teilautomatisierter Aufgaben durch Kooperation *personeller* und *maschineller Aufgabenträger* realisiert ist. Die Ressource Personal wird anhand ihrer *Aufbauorganisation* modelliert. Die Modellierung der Ressourcen *Anwendungssysteme* sowie *Maschinen und Anlagen* erfolgt anhand ihrer zugehörigen Spezifikationen [FeSi13, S. 197].

Entsprechend den Restriktionen des Untersuchungsobjekts der vorliegenden Arbeit sind die Aufstellung des Unternehmensplans sowie die Erstellung des Geschäftsprozessmodells nicht Gegenstand der Untersuchung (Abschnitt 1.2.1). Die Entwicklung von RESTful SOA erfolgt auf Basis fertiger („finaler“) SOM-Geschäftsprozessmodelle. Der Prozess der Gestaltung und des Managements von Geschäftsprozessmodellen sowie die Umsetzung dort definierter Ziele werden dabei nicht betrachtet. Auf der Ebene des Ressourcenmodells ist die Beschreibung von Aufgabenträgern ausschließlich auf die AwS-Spezifikation beschränkt. Für nähere Informationen zu den Themen Unternehmensplan und Gestaltung der Geschäftsprozessebene sei auf die weiterführende Literatur verwiesen (z. B. [FeSi13, S. 200 ff.], [HaWo12], [Hart11], [FeSi95]).

SOM-Vorgehensmodell (V-Modell)

Das *SOM-Vorgehensmodell*, oder kurz *V-Modell*, besteht aus drei Ebenen, die entsprechend den Modellebenen der Unternehmensarchitektur das Vorgehen der Modellierung von Unternehmensplan, Geschäftsprozessmodell und Ressourcenmodell beschreiben (Abbildung 4.2).

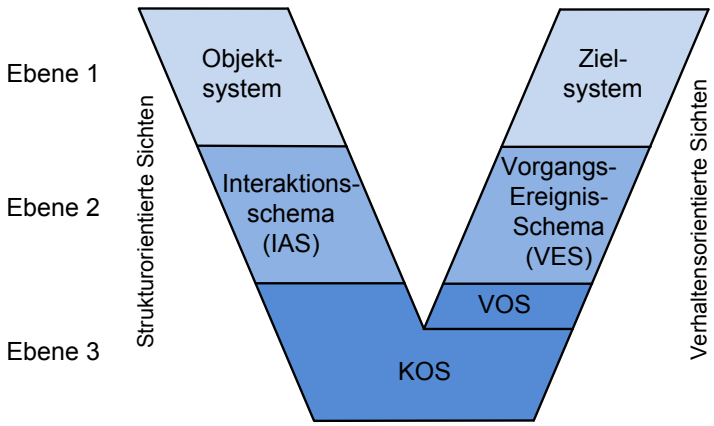


Abbildung 4.2: SOM-Vorgehensmodell [FeSi13, S. 198]

Jedes Teilmodellsystem wird anhand der Systemmerkmale *Verhalten* und *Struktur* in Form zweier Sichten spezifiziert. Die strukturorientierten Sichten werden dabei im linken, und die verhaltensorientierten Sichten im rechten Schenkel des V-Modells erfasst. Die Abstände zwischen den Schenkeln entsprechen Freiheitsgraden, die in Bezug auf die Gestaltung der Sichten einer Modellebene bestehen ([FeSi95, S. 212 f.], [FeSi13, S. 197 ff.]).

Die Modellierung der Sichten auf den drei Ebenen des Vorgehensmodells wird nachfolgend genauer erläutert:

- Das *Objektsystem* beschreibt die strukturorientierte Sicht auf den **Unternehmensplan**. Es beinhaltet die Abgrenzung der Diskurswelt und der relevanten Umwelt des betrieblichen Systems und definiert die zugehörigen Leistungsbeziehungen. Das *Zielsystem* spezifiziert mit Sach- und Formalzielen, Strategien sowie relevanten Rahmenbedingungen die verhaltensorientierten Aspekte dieser Ebene ([FeSi95, S. 213], [FeSi13, S. 198]).
- Die Struktursicht des **Geschäftsprozessmodells** wird mit dem *Interaktionsschema* (IAS) erfasst. Die im Unternehmensplan abgegrenzten Diskurs- und Umweltobjekte werden als betriebliche Objekte modelliert. Betriebliche Transaktionen verknüpfen betriebliche Objekte und realisieren die Koordination der Leistungserstellung und -übergabe. Das *Vorgangs-Ereignis-Schema* (VES) spezifiziert die zum IAS korrespondierende verhaltensorientierte Sicht des Geschäftsprozessmodells als ereignisgesteuerte Folge von Aufgaben. Die Modellierung des Geschäftsprozessmodells erfolgt durch schrittweise Verfeinerung von IAS und korrespondierendem VES. Ein integriertes Metamodell unterstützt die Modellierung korrespondierender Sichten und fördert damit die Herstellung von Konsistenz und Vollständigkeit zwischen diesen ([FeSi95, S. 213], [FeSi13, S. 198]).
- Die Ressource **Anwendungssystem** wird aus fachlicher Sicht im *konzeptuellen Objektschema* (KOS) sowie im *Vorgangsobjektschema* (VOS) spezifiziert. Das KOS erfasst eine Menge konzeptueller Objekttypen und deren Beziehungen. Das VOS besteht aus Vorgangsobjekttypen, welche die Durchführung von Aufgaben als Koordination einer Menge konzeptueller Objekttypen beschreiben. KOS und VOS spezifizieren demnach das Lösungsverfahren zur Durchführung automatisierter (Teil-)Aufgaben des Geschäftsprozesses ([FeSi95, S. 213], [FeSi13, S. 198 f.]).
  Der Modellierung von KOS und VOS liegt ebenfalls ein integriertes Metamodell zugrunde. Durch Einsatz eines Beziehungsmetamodells lassen sich die Modellelemente des Geschäftsprozessmodells und der fachlichen Spezifikation des AwS (Ressourcenmodell) zueinander in Beziehung setzen.

Ein Durchlaufen des V-Modells erfolgt idealtypisch von oben nach unten. Bei einer Abweichung von diesem Ideal, z. B. aus praktischen Gründen, sollte die konsistente Abstimmung der erstellten Modellebenen dennoch stets von oben nach unten vorgenommen werden. Zudem sind die Modellierungsergebnisse von Struktur- und Verhaltenssicht innerhalb einer Ebene sowie die Beziehungen dieser Sichten zu korrespondierenden Sichten benachbarter Ebenen aufeinander abzustimmen. Hierdurch wird die Konsistenz des (Gesamt-) Modellsystems sichergestellt ([FeSi13, S. 199], [FeSi95, S. 213]).

4.1.2 Modellierung von Geschäftsprozessen

Ein *Geschäftsprozess* (GP) wird im Allgemeinen als ereignisgesteuerter Ablauf einer Menge von Aktivitäten charakterisiert. Die Durchführung des Geschäftsprozesses bzw. seiner Aktivitäten wird dabei durch die Übernahme von Inputs ausgelöst. Als Ergebnis einer GP-Durchführung werden Outputs erzeugt und an die Umwelt bzw. die nachfolgende(n) Aktivität(en) übergeben. Der Zweck eines Geschäftsprozesses ist die Erstellung und Übergabe einer Leistung an Kunden. Zur Realisierung der GP-Durchführung werden Ressourcen zugeordnet und genutzt ([FeSi13, S. 200], [BeKa11, S. 5 f.], [Fran02], [Sche02, S. 3], [FeSi93]).

Ein Geschäftsprozess wird in der Literatur häufig auf seine verhaltensorientierten Eigenschaften und damit die ereignisgesteuerte Abfolge seiner Aktivitäten reduziert (z. B. [Rupp09, S. 189 ff.], [Grah08, S. 111 ff.], [KNS92]). In der SOM-Methodik wird diese Ablaufsicht um zwei strukturorientierte Sichten (Leistungssicht, Lenkungssicht) ergänzt und so eine umfassendere Perspektive auf Geschäftsprozesse eingenommen (folgende Auflistung nach [FeSi95, S. 214], [FeSi13, S. 200 f.]):

- **Leistungssicht**: Die *strukturorientierte Leistungssicht* beschreibt für einen Geschäftsprozess die Erstellung der betrieblichen Leistung und deren Übergabe an den beauftragenden Geschäftsprozess. Güter, Dienstleistungen oder Zahlungen sind mögliche Formen einer betrieblichen Leistung.
- **Lenkungssicht**: Die Koordination der betrieblichen Objekte, die an der Leistungserstellung und -übergabe beteiligt sind, wird in der *strukturorientierten Lenkungssicht* erfasst. Eine Koordination erfolgt nach dem *Verhandlungsprinzip* (nicht-hierarchische Koordination) oder dem *Regelungsprinzip* (hierarchische Koordination).
- **Ablaufsicht**: Der ereignisgesteuerte Ablauf der Aufgaben (Aktivitäten) eines Geschäftsprozesses wird in der *verhaltensorientierten Ablaufsicht* beschrieben. Dazu ist jede Aufgabe einem betrieblichen Objekt der Struktursicht zugeordnet und wird im Rahmen des GP-Ablaufs als Vorgang durchgeführt.

Die Bildung der strukturorientierten Leistungs- und Lenkungssicht auf einen Geschäftsprozess erfolgt in der SOM-Methodik im IAS. Das VES beschreibt die verhaltensorientierte Ablaufsicht [FeSi13, S. 201]. Die methodische Basis für die Modellierung von SOM-GPM bilden das Konzept des *betrieblichen Objekts* sowie die *transaktionsorientierte Koordination*. Die Zuordnung von Ressourcen zu den Aufgaben des SOM-GPM erfolgt auf der dritten Modellebene (Abschnitt 4.1.3).

Das Konzept des betrieblichen Objekts

Das betriebliche Objekt ist der konzeptuelle Kernbaustein in SOM-Geschäftsprozessmodellen. In Abstimmung mit der übergeordneten Metapher der SOM-Methodik wird die Bildung eines verteilten Systems als Menge autonomer, lose gekoppelter Objekte gesehen (Abschnitt 4.1.1).

Ein betriebliches Objekt besteht aus einer Menge von Aufgaben, die inhaltlich „*zusammengehörige Sach- und Formalziele verfolgen und auf einem gemeinsamen Aufgabenobjekt durchgeführt werden*“ [FeSi13, S. 203]. Betriebliche Objekte bauen somit auf dem Modell der

betrieblichen Aufgabe auf (Abschnitt 2.1.3). In der SOM-Methodik wird das allgemeine Aufgabenkonzept dahingehend erweitert, dass ein betriebliches Objekt eine Menge inhaltlich zusammengehöriger Aufgaben eines Geschäftsprozesses kapselt. Die Koordination mehrerer Objekte erfolgt durch den Austausch von Transaktionen. Ein betriebliches Objekt ist demnach autonom und lose gekoppelt [FeSi13, S. 203 ff.]. Das objektorientierte Konzept des betrieblichen Objekts zeigt Abbildung 4.3.

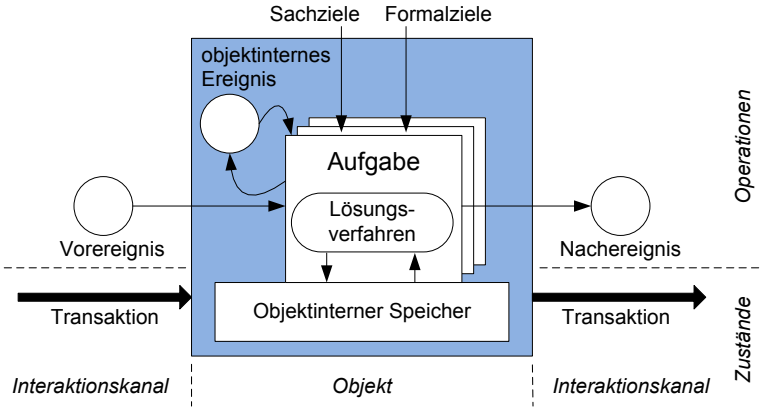


Abbildung 4.3: Objektorientiertes Konzept betrieblicher Objekte [FeSi13, S. 203]

Die *Aufgaben* eines betrieblichen Objekts verfolgen zusammengehörende *Sach- und Formalziele* und arbeiten auf einem gemeinsamen *Aufgabenobjekt*. Das Aufgabenobjekt umfasst die Attribute des *objektinternen Speichers* sowie Attribute der gesendeten oder empfangenen Lenkungs- und Leistungsflüsse. Über die Zustände des objektinternen Speichers sind die Aufgaben eines Objekts eng miteinander gekoppelt. Das *Lösungsverfahren* spezifiziert die Aufgabendurchführung durch Manipulation des Aufgabenobjekts. Dieses wird dabei von einem Vor- in einen Nachzustand überführt. Die Durchführung betrieblicher Aufgaben lösen Vorereignisse aus. *Vor- und Nachereignisse* beschreiben Reihenfolgebeziehungen zwischen den Aufgaben unterschiedlicher betrieblicher Objekte. Die Aufgaben eines betrieblichen Objektes können durch *objektinterne Ereignisse* verknüpft und dadurch in eine logische Ablaufbeziehung gebracht werden ([FeSi13, S. 203], Abschnitt 2.1.3).

Eine *Transaktion* stellt einen Kommunikationskanal zwischen betrieblichen Objekten zum Transport von Leistungspaketen bzw. Lenkungsnachrichten bereit. Leistungspakete und Nachrichten sind an Vor- und Nachereignisse gebunden. Die Kommunikation erfolgt auf Basis eines vereinbarten Kommunikationsprotokolls [FeSi13, S. 204]. Betriebliche Transaktionen sind atomar, d.h. sie werden gemäß den Eigenschaften einer Datenbanktransaktion nach dem Prinzip „Alles-oder-Nichts“ ausgeführt (z. B. [Unla13]).

Transaktionsorientierte Koordination lose gekoppelter Objekte

SOM-Geschäftsprozessmodelle werden in Form eines verteilten Systems modelliert, das aus einer Menge autonomer und lose gekoppelter betrieblicher Objekte besteht. Die Koordination der betrieblichen Objekte basiert auf dem Konzept der Transaktion und wird als *transaktionsorientierte Koordination* bezeichnet. Nach dem Client-Server-Prinzip können betriebliche Objekte in der Rolle eines Client-Objekts eine Leistungserstellung und -übergabe beauftragen, oder als Server-Objekt eine Leistung erbringen ([FeSi13, S. 204 ff.], [FeSi95, S. 217]). Zwischen

unterschiedlichen Objekten werden Leistungspakete und Lenkungsnachrichten in Form von Transaktionen ausgetauscht. Sie spezifizieren dadurch das Lösungsverfahren zur Erfüllung der Unternehmensaufgabe aus Struktursicht. Eine Verfeinerung der Koordination erfolgt in SOM grundsätzlich nach dem Verhandlungsprinzip oder Regelungsprinzip und resultiert in der Zerlegung von Objekten und Transaktionen.

Das *Verhandlungsprinzip* beschreibt die nicht-hierarchische Koordination zweier gleichrangiger betrieblicher Objekte beim Austausch eines Leistungspakets oder einer Lenkungsnachricht. Nach dem Verhandlungsprinzip werden Transaktionen phasenorientiert in Anbahnungs- (A), Verhandlungs- (V) und Durchführungstransaktion (D) zerlegt. Diese Transaktionen stehen in einer sequentiellen Reihenfolgebeziehung zueinander und werden phasenorientiert durchlaufen (Aufzählung nach [FeSi13, S. 205], [FeSi95, S. 217]):

- In der *Anbahnungsphase* lernen sich die Transaktionspartner kennen und tauschen Informationen über Leistungen aus. Diese Phase kann entfallen, wenn Transaktionspartner und Leistungen bereits bekannt sind.
- Der Leistungsaustausch wird zwischen den Transaktionspartnern in der *Vereinbarungsphase* festgelegt. Diese Phase endet mit der beidseitigen Verpflichtung der beteiligten Partner zum Leistungsaustausch.
- In der *Durchführungsphase* wird der vereinbarte Leistungsaustausch durchgeführt. Nach Beendigung dieser Phase ist der Leistungsaustausch abgeschlossen. Bezieht sich dieser auf bereits früher festgelegte Vereinbarungen zwischen Transaktionspartnern, so kann neben der Anbahnungs- auch die Vereinbarungsphase entfallen.

Nach dem *Regelungsprinzip* wird ein betriebliches Objekt in ein Lenkungsobjekt, ein Leistungsobjekt und eine Steuerungstransaktion (S) sowie in eine optionale Kontrolltransaktion (K) zerlegt [FeSi95, S. 217]. Damit liegt ein geregeltes System vor (vgl. [FeSi13, S. 27 ff.]):

- Das *Lenkungsobjekt* (*Regler*) umfasst alle Aufgaben zur Planung der Erstellung und Übergabe der betrieblichen Leistung und steuert hierzu das Leistungsobjekt hierarchisch in Form einer *Steuerungstransaktion* ([FeSi95, S. 217], [FeSi13, S. 205]).
- Ein *Leistungsobjekt* (*Regelstrecke*) besteht aus den Aufgaben zur Durchführung der Leistungserstellung und meldet optional ihren Ausführungszustand an die Regelstrecke in Form einer *Kontrolltransaktion* ([FeSi95, S. 217], [FeSi13, S. 205]).

Die Durchführung der Leistungserstellung visualisiert Abbildung 4.4 in allgemeiner Form. Die Darstellung zeigt zwei betriebliche Objekte, die sich transaktionsorientiert koordinieren. Die Transaktion wird nach dem Verhandlungsprinzip in A-, V- und D-Transaktion zerlegt. Das leistungserbringende Server-Objekt (links) wird nach dem Regelungsprinzip zudem in ein Leistungs- und ein Lenkungsobjekt untergliedert, die sich durch S- und K-Transaktion hierarchisch koordinieren.

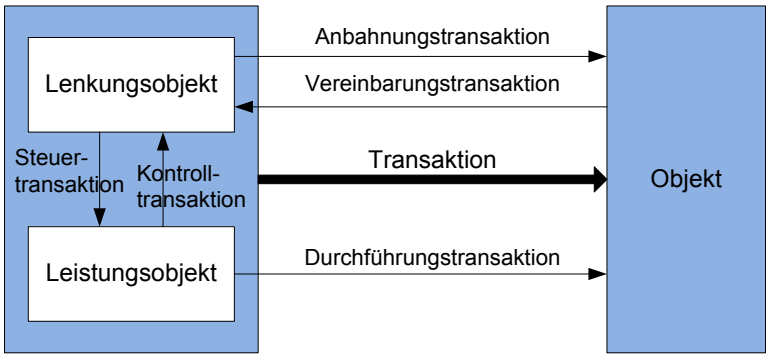


Abbildung 4.4: Transaktionsorientierte Koordination betrieblicher Objekte [FeSi13, S. 204]

Nach der Anbahnung beauftragt das (rechts stehende) Client-Objekt mittels Vereinbarungstransaktion das Server-Objekt mit der Leistungserbringung. Nach dem Empfang der Vereinbarung durch das Lenkungsobjekt weist dieses das Leistungsobjekt über eine Steuertransaktion zur Leistungserbringung an. Nach erfolgreicher Leistungserstellung wird diese mittels Durchführungstransaktion dem Client-Objekt übergeben. Anschließend wird der Abschluss der Beauftragung über eine Kontrolltransaktion an das Lenkungsobjekt gemeldet (z. B. [FeSi95, S. 217], [FeSi13, S. 204 f.]).

Geschäftsprozesse werden in SOM nach dem Konzept der hierarchischen Zerlegung modelliert. Ausgehend von einem initialen Modell wird auf Basis der vorgestellten Koordinationsprinzipien eine schrittweise Verfeinerung des Geschäftsprozessmodells durch Zerlegung der betrieblichen Objekte und Transaktionen durchgeführt. Alle zulässigen Zerlegungsschritte für betriebliche Objekte und Transaktion sind in Form von *Ersetzungsregeln* definiert. Eine detaillierte Erläuterung des hierarchischen Zerlegungskonzepts findet sich in [FeSi13, S. 207 ff.].

Integriertes Metamodell der SOM-Geschäftsprozessmodellebene

Die Modellierung der Struktur- und Verhaltenssicht von SOM-Geschäftsprozessmodellen basiert auf einem integrierten Metamodell. Wie bereits im obigen Abschnitt ausgeführt, wird die strukturorientierte Sicht eines Geschäftsprozesses im IAS und die korrespondierende verhaltensorientierte Sicht im VES modelliert ([FeSi06, S. 351 ff.], [FeSi95, S. 206]).

Das *SOM-Metamodell* definiert die Sprache zur Geschäftsprozessmodellierung als Menge von Metaobjekten und Beziehungen zwischen den Metaobjekten. Die Metaobjekte eines SOM-GPM sind betriebliches Objekt, betriebliche Transaktion, Aufgabe, Leistung sowie objektinternes Ereignis und externes Umwelt-Ereignis. Das integrierte SOM-Metamodell der Geschäftsprozessebene zeigt Abbildung 4.5. Es basiert auf dem Meta-Metamodell nach Sinz (Abschnitt 2.1.5). Die Sichten IAS und VES sind als Projektionen auf das integrierte Metamodell definiert [FeSi13, S. 218 f.].

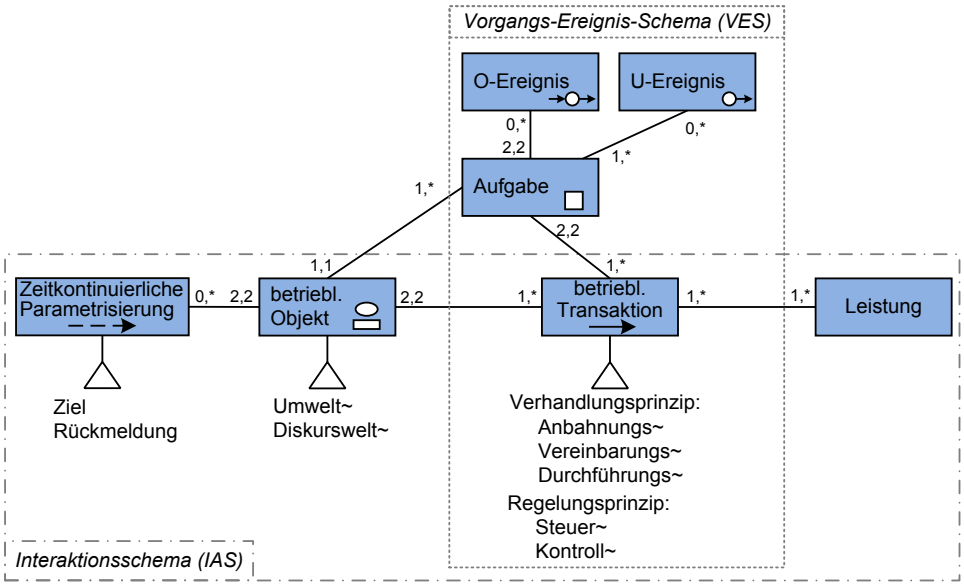


Abbildung 4.5: SOM-Metamodell zur Modellierung von Geschäftsprozessen (in Anlehnung an [FeSi13, S. 219])

Ein *betriebliches Objekt* umfasst eine bis beliebig viele Aufgaben und sendet oder empfängt eine bis beliebig viele Transaktionen. Eine *Aufgabe* ist genau einem betrieblichen Objekt zugeordnet und steht mit einer bis beliebig vielen betrieblichen Transaktionen in Beziehung. Betriebliche Objekte sind Diskurswelt- (Rechteck) oder Umweltobjekte (Ellipse). Die *betriebliche Transaktion* ist zur nicht-hierarchischen Koordination nach dem Verhandlungsprinzip und zur hierarchischen Koordination nach dem Regelungsprinzip zerlegbar. Eine betriebliche Transaktion verbindet immer zwei Aufgaben zweier unterschiedlicher Objekte. Jede *Leistung* wird von einer bis beliebig vielen Transaktionen übertragen oder koordiniert. Genau zwei Aufgaben eines Objekts können durch *objektinterne Ereignisse* (O-Ereignis) verknüpft werden. *Umweltereignisse* stellen externe, nicht in den Transaktionen erfasste Ereignisse dar [FeSi13, S. 219].

Ziele und Rückmeldungen dienen der *zeitkontinuierlichen Parametrisierung* von Aufgaben. Die Ziele eines betrieblichen Objekts sind mit jeder seiner Aufgaben verknüpft [FeSi13, S. 219]. Im weiteren Verlauf wird der Baustein der zeitkontinuierlichen Parametrisierung von betrieblichen Objekten nicht näher betrachtet, da sich die Ziele von Aufgaben nicht direkt in der Spezifikation der AwS-Struktur widerspiegeln. In der vorliegenden Arbeit werden Ziele lediglich durch ihre Realisierung im Lösungsverfahren von Aufgaben berücksichtigt. In diesem Zusammenhang sei auf die Untersuchung von HARTMANN ET AL [HTW13] zur Spezifikation funktionaler und nicht-funktionaler Anforderungen in Geschäftsprozessmodellen verwiesen. Einen Überblick über die Modellierung strategischer Ziele in gängigen Geschäftsprozessmodellen geben HARTMANN und WOLF [HaWo12].

ERLÄUTERNDES BEISPIEL HANDELSBETRIEB

Abbildung 4.6 zeigt das einfache SOM-GPM eines Handelsbetriebs (in Anlehnung an [FeSi95, S. 217 f.]). Der Handel bahnt die Leistungserstellung mittels Versand von Werbung (Preislisten der Produkte) zum Kunden an (A:Werbung). Anschließend bestellt der Kunde die gewünschten

Produkte (V:Auftrag). Mit dem Empfang der Produktlieferung ist die Durchführung der betrieblichen Leistungserstellung und -übergabe (D:Lieferung) abgeschlossen.

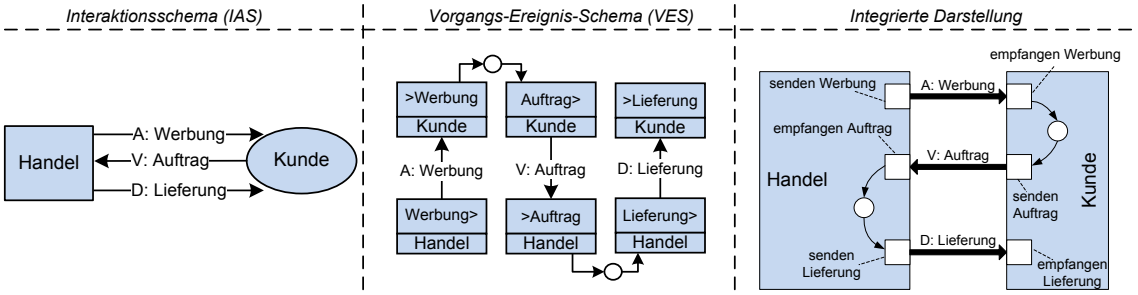


Abbildung 4.6: SOM-Geschäftsprozessmodell eines Handelsbetriebs

Die Struktur des Geschäftsprozesses zeigt das IAS (linker Teil der Abbildung). Der ereignisgesteuerte Ablauf von Aufgaben und damit das GP-Verhalten werden im VES erfasst (mittlerer Teil). Die Aufgaben beschreiben das Lösungsverfahren zum Versand (*Transaktion>*) oder Empfang (*>Transaktion*) einer Transaktion. Zur Veranschaulichung des Konzepts der Sichtenbildung in der SOM-Geschäftsprozessmodellierung wird das Modell zudem in integrierter Form dargestellt (rechter Teil). Aus Gründen der Bewältigung von Komplexität in detaillierten SOM-GPMs wird die integrierte Darstellungsform jedoch i. d. R. nicht verwendet.

4.1.3 Fachliche Spezifikation von Anwendungssystemen

Das Ressourcenmodell der SOM-Unternehmensarchitektur umfasst die personellen und maschinellen Aufgabenträger zur Durchführung der Aufgaben in Geschäftsprozessen. Die Ressource *Anwendungssystem* als maschineller Aufgabenträger für die informationsverarbeitenden Aufgaben (IS-Aufgaben) von Geschäftsprozessen steht im Fokus dieser Arbeit. Die fachliche AwS-Spezifikation erfolgt im Anwendungsmodell auf der fachlichen Ebene der Systementwicklung (Abschnitt 2.2.2).

Voraussetzung für die Spezifikation von AwS ist ein hinreichend detailliert modelliertes SOM-GPM. Es bildet den Ausgangspunkt für die Bestimmung der Automatisierung des Geschäftsprozesses sowie die Abgrenzung des Anwendungssystems. Darauf aufbauend erfolgt die Spezifikation des Anwendungsmodells ([FeSi13, S. 220 ff.], [FeSi06, S. 359 f.]).

AUTOMATISIERUNG VON GESCHÄFTSPROZESSEN

Die Beziehung zwischen Anwendungssystem und Geschäftsprozessmodell wird in der SOM-Methodik anhand der Automatisierung von Aufgaben und Transaktionen beschrieben, die das AwS in automatisierter oder teilautomatisierter Form durchführt [FeSi13, S. 220 f.].

Bei der *Automatisierung von Aufgaben* wird zwischen vollautomatisierten, teilautomatisierten und nicht-automatisierten IS-Aufgaben unterschieden (Abschnitt 2.1.3). Eine v*ollautomatisierte* Aufgabe wird vollständig von einem Anwendungssystem durchgeführt. Die Durchführung von *teilautomatisierten* Aufgaben erfolgt in kooperativer Form zwischen den Aufgabenträgern Anwendungssystem und Person. *Nicht-automatisierte* Aufgaben werden von personellen Aufgabenträgern vollständig übernommen [FeSi13, S. 55 f. und 220 f.].

Bei der *Automatisierung von Transaktionen* eines Geschäftsprozesses wird lediglich zwischen den Graden *voll-automatisiert* und *nicht-automatisiert* unterschieden. Dient das Kommunikationssystem dem Nachrichtenaustausch zwischen Computer und Computer (CCK) oder Mensch und Computer (MCK), so ist eine Transaktion vollautomatisiert. Dient das Kommunikationssystem dem Nachrichtenaustausch zwischen Mensch und Mensch, so ist die Transaktion nicht-automatisiert [FeSi13, S. 220 f.].

ABGRENZUNG DES ANWENDUNGSSYSTEMS

Ein Geschäftsprozess kann von einem oder mehreren Anwendungssystemen unterstützt werden. In der SOM-Methodik erfolgt die AwS-Abgrenzung in prozessorientierter Form. Dabei werden Anwendungssysteme grundsätzlich anhand der betrieblichen Objekte auf der GP-Ebene abgegrenzt und den zu automatisierenden GP-Aufgaben zugeordnet [FeSi13, S. 222]. Die Anwendung dieses Ansatzes basiert dabei auf der Annahme, dass beide Ebenen „homomorphe Strukturen“ aufweisen und somit eine Überführung durch eine metamodellbasierte Transformation erfolgen kann ([FeSi96], [FeSi13, S. 222]).

Die vorliegende Arbeit untersucht die modellbasierte Spezifikation von RESTful SOA als *alleiniges Anwendungssystem* zur Unterstützung von SOM-Geschäftsprozessmodellen. Die Abgrenzung und Zuordnung mehrerer AwS wird aus diesem Grund nicht tiefergehend betrachtet. Nähere Informationen zu diesem Thema finden sich in den Arbeiten von SINZ und TEUSCH, die ein Vorgehen zum fachlichen Entwurf partieller SOA in bestehenden AwS-Landschaften vorschlagen ([TeSi12], [KrSi11]). Darüber hinaus sei auf die weiterführende Literatur zur Spezifikation von AwS in der SOM-Methodik verwiesen (z. B. [FeSi13, S. 220 ff.], [FeSi06], [FeSi97]).

METAMODELL ZUR FACHLICHEN SPEZIFIKATION VON ANWENDUNGSSYSTEMEN

Nach der Bestimmung der Automatisierungsgrade von Aufgaben und Transaktionen sowie der Abgrenzung des zu automatisierenden Geschäftsprozessausschnitts erfolgt die Spezifikation des Anwendungssystems im fachlichen Anwendungsmodell. Hierzu wird das Anwendungsmodell zunächst aus dem SOM-Geschäftsprozessmodell initial abgeleitet und anschließend verfeinert (siehe Abschnitt 5.5).

In der SOM-Methodik wird das zu entwickelnde AwS als objektorientiertes und objektintegriertes verteiltes System spezifiziert, welches aus einer Menge lose gekoppelter Objekttypen besteht. Diese kooperieren im Hinblick auf die Erreichung gemeinsamer Ziele. Die Koordination zwischen den Objekttypen erfolgt durch den Austausch von Nachrichten [FeSi13, S. 222 f.].

Abbildung 4.7 zeigt das SOM-Metamodell zur *fachlichen Spezifikation von Anwendungssystemen*. Eine objektorientierte AwS-Spezifikation besteht in SOM aus einer Menge von *Objekttypen*, die miteinander über Beziehungen verknüpft sind. Ein Objekttyp und seine Instanzen bilden eine *Klasse*. Eine *Beziehung* verbindet immer genau zwei Objekttypen. Durch die Zuordnung von *Attributen* und *Operatoren* werden die Eigenschaften von Objekttypen beschrieben. Eine Konkretisierung der *Beziehungen* erfolgt anhand ihrer Spezialisierung als *Generalisierungs-* (*is_a*), *Assoziations-* (*interacts_with*) oder *Aggregations-Beziehung* (*is_part_of*) [FeSi13, S. 224 f. und S. 233 f.].

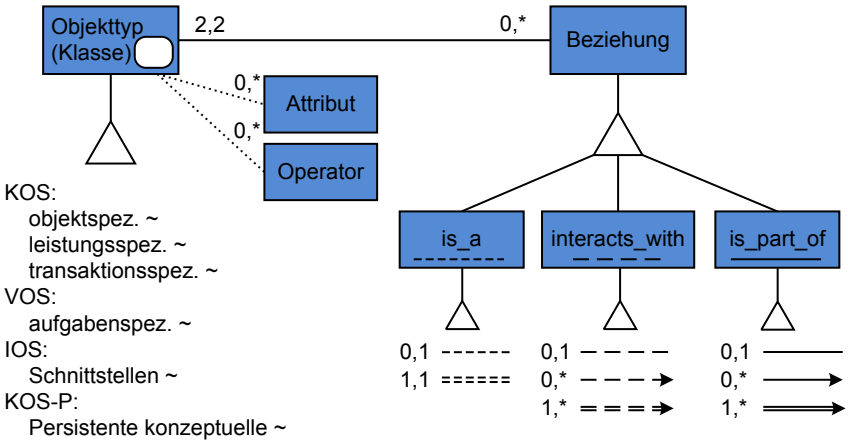


Abbildung 4.7: Metamodell zur fachlichen Spezifikation von Anwendungssystemen (in Anlehnung an [FeSi13, S. 233])

Das Anwendungssystem wird in der SOM-Methodik in Form von vier Modellschemata spezifiziert. Jedes Schema enthält eine Menge zusammengehöriger Objekttypen und beschreibt die Realisierung eines definierten Funktionsbereichs des AwS. Auf Basis der Spezialisierung von Objekttypen ergeben sich folgende vier Objektschemata (Aufzählung nach [FeSi13, S. 234]):

- Das **konzeptuelle Objektschema** (**KOS**) „*umfasst objektspezifische, leistungsspezifische und transaktionsspezifische Objekttypen, die mit den Komponenten betriebliches Objekt, Leistung und Transaktion eines SOM-Geschäftsprozessmodells korrespondieren*“ [FeSi13, S.234]. Die Objekttypen dieses Schemas werden als *konzeptuelle Objekttypen* (KOT) bezeichnet.
- Das **Vorgangsobjektschema** (**VOS**) besteht aus *aufgabenspezifischen Objekttypen*, die mit den Aufgaben eines VES korrespondieren. Aufgabenspezifische Objekttypen werden auch als *Vorgangsobjekttypen* (VOT) bezeichnet und spezifizieren das Lösungsverfahren der Aufgabendurchführung als Zusammenwirken einer Menge von KOTs.
- Das **Interface-Objektschema** (**IOS**) umfasst *Schnittstellen-* bzw. *Interface-Objekttypen* (IOT), die die Schnittstellen zu personellen und maschinellen Aufgabenträgern spezifizieren. Sie beschreiben die Oberfläche und die verwendeten Protokolle zur Interaktion mit Aufgabenträgern. Die Zuordnung und Spezifikation von IOTs ist eng mit der Entwicklung der VOTs abgestimmt, die mit Aufgabenträgern der Umwelt über IOTs in Beziehung stehen.
- Das **konzeptuelle Objektschema persistenter Klassen** (**KOS-P**) umfasst diejenigen KOTs, deren Zustände über den Zeitraum der Aufgabendurchführung hinaus zu persistieren sind.

In der SOM-Methodik wird die Spezifikation von AwS nach den Verantwortlichkeiten der Objekttypen in einem Schichtenmodell strukturiert. Nach dem Konzept des ADK-Strukturmodells ([FeSi13, S. 235] und Abschnitt 2.2.4.3) erfolgt die Realisierung von

- Anwendungsfunktionen durch VOS und KOS,
- Kommunikation (K_P, K_M) durch IOS, und der
- Datenhaltung durch KOS-P.

Die Interaktionen zwischen den Instanzen der Objekttypen sind im Allgemeinen ebenfalls durch ihre Verantwortlichkeiten bestimmt. Eine verbreitete Realisierungsform sieht die Steuerung der AwS-Ausführung durch VOTs vor [FeSi13, S. 234]. Das zulässige Zusammenwirken zwischen den Objekttypen nach diesem Prinzip zeigt Abbildung 4.8.

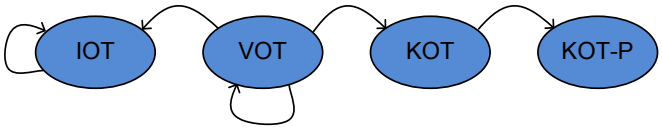


Abbildung 4.8: Aufrufbeziehungen zwischen den Instanzen der Objekttypen (in Anlehnung an [FeSi13, S. 235])

Die Ausführung des AwS steuert ein verteiltes System von VOTs. Ein VOT ruft dabei die ihm zugeordneten IOTs, VOTs und KOTs auf und koordiniert den Ablauf des von ihm kontrollierten Teils der Anwendungsfunktionalität. IOTs realisieren die Schnittstellen zur Kommunikation mit externen Aufgabenträgern. KOTs kapseln die Zustände von Aufgaben (objektinterner Speicher) und Transaktionen, sowie Operationen zur Manipulation dieser Zustände. Sie spezifizieren demnach die (domänenspezifische) Geschäftslogik der Anwendung. Die Persistierung der Zustände eines KOT erfolgt durch einen KOT-P. Eine direkte Kommunikation zwischen KOTs oder zwischen KOT-Ps findet nicht statt ([FeSi13, S. 234 f.], [FeSi06, S. 362], [FeSi94, S. 4]).

Hinweise zur fachlichen Spezifikation von Anwendungssystemen

Die Anwendungsfunktionen (A-Teil), die den Kern des objektorientierten Anwendungssystems bilden, werden in der SOM-Methodik durch die Schemata *VOS* und *KOS* beschrieben. Bezüglich einer grafischen Darstellung der AwS-Spezifikation im fachlichen Anwendungsmodell werden in dieser Arbeit folgende vereinfachenden Festlegungen getroffen:

- Das IOS spezifiziert die Schnittstellen zur Kommunikation mit der Anwendungsumgebung (K-Teil) als Menge von IOTs, die jeweils ihren korrespondierenden VOTs zugeordnet sind. Die Schemata IOS und VOS werden zur übersichtlicheren Darstellung in einem gemeinsamen Schema erfasst. Somit ist die Beschreibung der Zuordnungsbeziehungen in einer Sicht möglich.
- Das KOS-P besteht aus den Objekttypen des Anwendungssystems, welche die Funktionalität der Persistierungsschicht realisieren und demnach die Datenhaltung (D-Teil) eines KOS repräsentieren. Deshalb erfolgt eine Spezifikation von KOT-Ps im Allgemeinen zusammen mit KOTs in einem gemeinsamen Schema. Auf die Darstellung des KOS-P als eigenes Schema wird im weiteren Verlauf verzichtet[19]. Die Funktionalität zur Realisierung der Datenhaltung erfasst das KOS.

Die Objektschemata IOS und VOS sowie KOS bilden den Ausgangspunkt für die softwaretechnische Spezifikation von betrieblichen Anwendungssystemen. Eine Entwicklungsmethodik zur softwaretechnischen Spezifikation von RESTful SOA auf Basis der SOM-Methodik wird in Kapitel 5 erarbeitet und vorgestellt.

[19] Die explizite Spezifikation eines KOS-P ist notwendig, wenn KOTs nicht vollständig oder nur Teile des KOS persistent verwaltet werden.

ERLÄUTERNDES BEISPIEL HANDELSBETRIEB (FORTSETZUNG)

Die Ausführungen zur fachlichen Spezifikation des AwS schließen mit dem Anwendungsmodell des Handelsbetriebs, das Abbildung 4.9 in vereinfachter Form zeigt. Das KOS des Handelsbetriebs (linke Seite) umfasst eine Menge konzeptueller Objekttypen, die aus der Leistung (Produktvertrieb), den betrieblichen Objekten (Handel, Kunde) sowie den Transaktionen (Produktinformation, Auftrag, Lieferung) des SOM-Geschäftsprozessmodells abgeleitet werden. Die Eigenschaften der KOTs *Kunde* und *Auftrag* sind bereits spezifiziert.

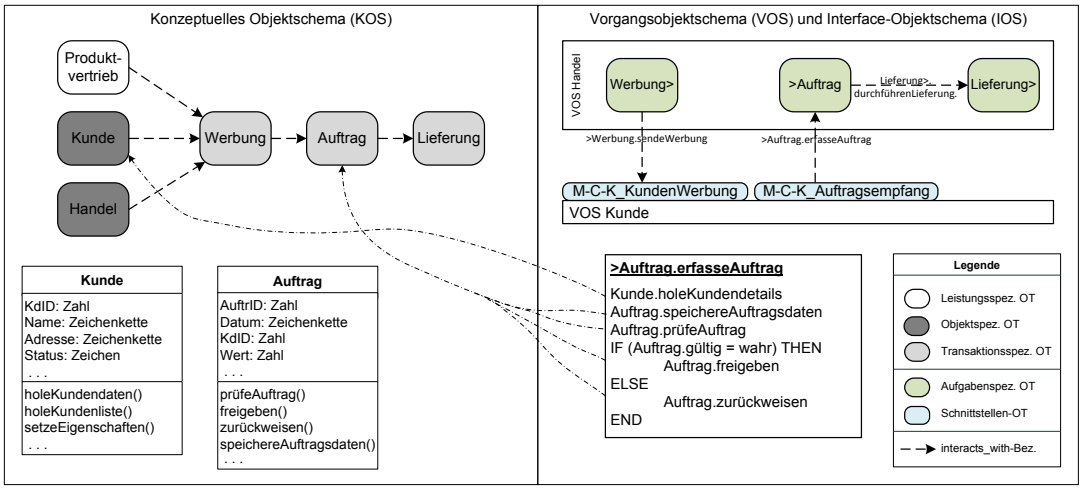


Abbildung 4.9: Konsolidiertes KOS und VOS/IOS am Beispiel eines Handelsbetriebs

VOS und IOS werden in einem gemeinsamen Schema visualisiert. Das VOS *Kunde* umfasst eine Menge von Vorgangsobjekttypen, die den Workflow zur Durchführung einer korrespondierenden Aufgabe des VES beschreiben. Der Workflow für die Auftragserfassung ist beispielhaft dargestellt. Zur Kommunikation mit den (nicht-automatisierbaren) Aufgaben des Kunden wird jedem VOT jeweils ein IOT für die MC-Kommunikation zugeordnet. Die Transaktion *Lieferung* wird im Beispiel nicht automatisiert.

4.2 Flexibilität in betrieblichen Systemen

Flexible SOM-Geschäftsprozessmodelle bilden den Ausgangspunkt für die modellbasierte Spezifikation von RESTful SOA. Im nachfolgenden Abschnitt wird der Flexibilitätsbegriff zunächst für *betriebliche Systeme* aus Sicht der Betriebswirtschaftslehre eingeführt (Abschnitt 4.2.1) und anschließend in Bezug auf das *betriebliche Informationssystem* als Betrachtungsgegenstand dieser Arbeit konkretisiert (Abschnitt 4.2.2). Auf dieser Basis wird der Flexibilitätsbegriff für *SOM-Geschäftsprozessmodelle* erarbeitet (Abschnitt 4.2.3).

4.2.1 Der Flexibilitätsbegriff in der Betriebswirtschaftslehre

Aus betriebswirtschaftlicher Sicht ist der Begriff *Flexibilität* bis heute nicht einheitlich definiert (z. B. [KaBl05b], [ShMo98], [Wolf89]). Vom lateinischen Wort „flexibilis“ abstammend, bedeu-

tet Flexibilität ursprünglich „biegsam“ und „geschmeidig“[20]. Dabei wird grundsätzlich zwischen zwei Wortbedeutungen unterschieden[21]: Im Sinne einer physikalischen Elastizität oder Biegsamkeit meint Flexibilität zum einen die Fähigkeit eines Körpers, sich als Reaktion auf externe Krafteinwirkung zu verformen und seinen Ausgangszustand nach Ende dieser Krafteinwirkung wieder einzunehmen. Zum anderen wird unter dem Begriff allgemein die Fähigkeit zu einem anpassungsfähigen, und damit flexiblen, Verhalten bei veränderlichen Umweltbedingungen verstanden (z. B. [MaBu90, S. 17], [Altr84, S. 605]).

Die Anfänge der Flexibilitätsforschung in der deutschsprachigen Betriebswirtschaftslehre liegen in den 20er Jahren des letzten Jahrhunderts (z. B. [Schm28], [Schm26] nach [KaBl05b, S. 6]). In dieser Zeit wurde Flexibilität vor allem im Kontext der Anpassung von Unternehmen an Marktänderungen untersucht, und meist als *Elastizität* bezeichnet. KALVERAM versteht unter „*Elastizität in der Betriebswirtschaftslehre [den] Grad der Anpassungsfähigkeit an die jeweilige Marktlage*“ ([Kalv31, S. 705] nach [Moos09, S. 57]). Diese Einschränkung auf Marktschwankungen wird in späteren Arbeiten aufgegeben und der Betrachtungsgegenstand auf die Bewältigung von Unsicherheiten im Umfeld betrieblicher Systeme erweitert (z. B. [Adam90]). Dieser thematische Schwerpunkt liegt auch der aktuellen betriebswirtschaftlichen Forschung zugrunde ([Moos09, S. 57], [KaBl05b, S. 6]).

Flexibilität von betrieblichen Systemen (z. B. Unternehmungen) wird somit meist im Sinne *evolutionärer Anpassung* verwendet [Burm02, S. 46 f.]. Dennoch existiert in der betriebswirtschaftlichen Literatur eine Vielzahl an Definitionen des Begriffs der Flexibilität[22]. Die unterschiedlichen Definitionen stimmen jedoch weitestgehend darin überein, dass sie Flexibilität als *„Anpassungs- und Änderungsfähigkeit in Bezug auf unterschiedliche Bedingungen […] verstehen*“ [KaBl05b, S. 8]. KALUZA und BLECKER definieren

> *„Flexibilität [als] die Eigenschaft eines Systems, proaktive oder reaktive sowie zielgerichtete Änderungen der Systemkonfiguration zu ermöglichen, um die Anforderungen von sich verändernden Umweltbedingungen zu erfüllen*“ [KaBl05b, S. 9].

Nach diesem weit gefassten und allgemeineren Verständnis wird unter Flexibilität somit nicht nur eine reaktive Anpassung des Systems zur Vermeidung von Risiken, sondern auch die antizipative Anpassung zur (aktiven) Nutzung von Chancen gesehen. Den Betrachtungsgegenstand bilden normalerweise betriebliche Systeme (Abschnitt 2.1.1).

Schwerpunkte der betriebswirtschaftlichen Flexibilitätsforschung sind vor allem die Bereiche der Flexibilität in (der Gestaltung von) Produktion und Organisation betrieblicher Systeme durch die Anpassung von Strukturen und Abläufen, sowie das Themenfeld der Planung und

---

[20] Stichwort „Flexibilität“ (Deutsch – Latein) auf PONS Online-Wörterbuch unter http://de.pons.com/ (Abruf am 25. März 2015).

[21] Stichwort: „Flexibilität“ auf Duden-Online unter http://www.duden.de/ (Abruf am 25. März 2015)

[22] Eine Untersuchung der englischsprachigen Literatur nach SHEWCHUK und MOODIE [ShMo98] ergab über 70 Definitionen von Flexibilität. Eine Vielzahl von Ansätzen zur Bestimmung des Flexibilitätsbegriffs werden in der Arbeit von WOLF [Wolf89] strukturiert dargestellt. Für einen tiefergehenden Einblick in unterschiedliche Flexibilitätsdefinitionen sei auf diese Arbeiten verwiesen.

Entscheidung unter Unsicherheit (z. B. [BuMa08], [KaBl05a], [ScWi04], [Klum02], [Adam90], [Hopf89], [Behr85], [Jaco74], [Meff68], [Jaco67]).

Zur Sicherung der Überlebensfähigkeit eines Unternehmens wird Flexibilität als einer der *strategischen Erfolgsfaktoren*[23] in der betriebswirtschaftlichen Forschung definiert (z. B. [KaBl05b, S. 4], [Meff85, S. 121 ff.]). Demnach ist die Fähigkeit, sich an dynamische Umweltänderungen rasch anzupassen und die bestehenden Unsicherheiten sowie hohe Komplexität der Umwelt beherrschen zu können, ein zentraler Wettbewerbsfaktor für die Erzielung von unternehmerischem Erfolg in betrieblichen Systemen (z B. [Moos09], [KaBl00], [Meff85]).

4.2.2 Flexibilität in betrieblichen Informationssystemen

Das Verständnis von Flexibilität im Sinne der Anpassungsfähigkeit (vgl. Abschnitt 4.2.1) wird in diesem Abschnitt deshalb auf betriebliche Informationssysteme als zentraler Betrachtungsgegenstand der Wirtschaftsinformatik übertragen. Allgemein lässt sich Flexibilität somit als *Anpassungsfähigkeit eines Informationssystems in Bezug auf eine sich wandelnde Umwelt oder Diskurswelt* beschreiben. Die Flexibilität betrieblicher Informationssysteme (IS-Flexibilität) wird im Folgenden auf Aufgaben- und Aufgabenträgerebene genauer untersucht.

Flexibilität auf der Aufgabenträgerebene von Informationssystemen

Die Betrachtung von IS-Flexibilität fokussiert in vielen Arbeiten meist die *Ebene der Aufgabenträger* (z. B. [TuLa05], [ByTu00], [Dunc95]). Sie wird dabei zum einen als Anpassungsfähigkeit der Anwendungssysteme und zugrundeliegenden IT-Infrastruktur verstanden, um Veränderungen in den sich wandelnden Geschäftsprozessen oder anderen IS-relevanten Prozessen zu unterstützen [Dunc95, S. 44]. Zum anderen bezieht sich der Begriff auf die Strukturierung der Aufgabenträger selbst. Ferstl und Sinz beschreiben die Trennung von Innen- und Außensicht als Merkmal flexibler IT-Systemarchitekturen, um programmbasierte Nutzermaschinen unabhängig von den eingesetzten Basismaschinen realisieren zu können [FeSi13, S. 320]. Weitere Untersuchungen schließen darüber hinaus personelle Aufgabenträger des IS ein, welche die Verwaltung sowie Anpassung der Aufgabenträgerebene durchführen (z. B. [TuLa05], [ByTu00, S. 192]).

Auf der Aufgabenträgerebene von Informationssystemen ist Flexibilität ein wichtiger **Erfolgsfaktor** zur *Verbesserung der Wettbewerbsfähigkeit* betrieblicher Systeme. Vor allem wird eine engere Abstimmung zwischen IT- und Geschäftsebene sowie die effiziente Anpassung von Aufgabenträgern an Änderungen auf der Aufgabenebene unterstützt (z. B. [CRB03, S. 20 ff.], [TuLa05], [BER+10]).

Als **Kriterien für Flexibilität** auf der Aufgabenträgerebene werden die *Verknüpfbarkeit*, *Kompatibilität* und *Modularität* von Komponenten der IT-Infrastruktur genannt ([Dunc95, S. 47 f. und S. 52], [ByTu00, S. 191 f.]). Ein flexibles IT-System unterstützt besonders die *Wiederverwendbarkeit* und *verteilte Nutzbarkeit* seiner Ressourcen [Dunc95, S. 42 f.].

[23] Strategische Erfolgsfaktoren sind Kosten, Qualität, Zeit, Erzeugnisvielfalt, Service sowie Flexibilität ([KaBl05b, S. 4], [Meff85, S. 121 ff.]).

Auf Basis der genannten Kriterien lässt sich die *RESTful SOA* als *flexible Systemarchitektur* charakterisieren. Durch die Bereitstellung lose gekoppelter Ressourcen mit einheitlicher Schnittstelle und der Nutzung von HTTP als Standard-Anwendungsprotokoll werden die Kompatibilität und Modularität von Systemkomponenten unterstützt. Der Einsatz von URIs und der Aufbau der Hypermedia zielen auf die Verknüpfbarkeit von Komponenten. Durch die Veröffentlichung einer implementierungsunabhängigen Schnittstelle erfolgt zudem eine Trennung der Innen- und Außensicht der bereitgestellten Komponenten von SOA.

FLEXIBILITÄT AUF DER AUFGABENEBENE VON INFORMATIONSSYSTEMEN

Eine Einschränkung der Betrachtung von IS-Flexibilität auf die Aufgabenträgerebene allein würde nur einen Teil der vorliegenden Flexibilität erfassen, da deren Ursachen häufig in veränderten Anforderungen auf der *Aufgabenebene* begründet sind. Eine Untersuchung von IS-Flexibilität der Aufgabenebene erfolgte im Rahmen des Forschungsverbundes *forFLEX* [SBB+11]. Auf Basis einer Literaturanalyse von IS-Flexibilitätsdefinitionen monieren WAGNER ET AL ([WSL+11a], [WSL+11b]), dass existierende Definitionen von spezifischen IS-Merkmalen abstrahieren und den Begriff zu generisch formulieren. Zudem werden durch die Einschränkung des Gegenstandsbereichs, z. B. IT-Infrastruktur- oder Technologieflexibilität, oder der Flexibilitätsarten, z. B. Strategie- oder Strukturflexibilität, wichtige IS-Merkmale nicht berücksichtigt [WSL+11a, S. 82 ff.]. Die Autoren schlagen zur Vermeidung dieser Defizite eine Untersuchung von IS-Flexibilität aus *systemtheoretischer Sicht* vor und nutzen die allgemeine Systemtheorie als theoretische Grundlage für ihre eigene Definition:

> „*Flexibilität ist die Fähigkeit eines Systems, auf System- oder Umweltveränderungen unter Berücksichtigung gegebener Ziele durch Anpassung von Struktur und/oder Verhalten zu reagieren oder sie zu antizipieren“* [WSL+11a, S. 88].

Nach diesem Verständnis sind sowohl externe Umwelt-, als auch interne Systemveränderungen Gründe für eine Anpassung des Informationssystems (IS-Flexibilität). Dabei kann die Anpassung als Reaktion auf Veränderungen oder durch deren Antizipation erfolgen. Zur Bewältigung von Flexibilität entwickeln Systeme ihre Struktur und/oder ihr Verhalten weiter ([BBF+11, S. 2 f.], [WSL+11a, S. 87 f.]).

Die eingeführte Systemtheorie-basierte Definition von Flexibilität ist Ausgangspunkt für eine Ausarbeitung des Flexibilitätsbegriffs der Aufgabenebene von Informationssystemen durch die Autoren, dessen methodische Grundlage das Aufgabenmodell bildet (Abschnitt 2.1.3). Grundsätzlich ist das Auftreten von Flexibilität in jedem Bestandteil der betrieblichen Aufgabe (Aufgabenmerkmalen) möglich. Die hierbei unterschiedenen *Flexibilitätsarten* ergeben sich aus Struktur und Verhalten der Aufgabe ([WSL+11b, S. 88 ff.], [WSL+11b, S. 811 f.]).

Verhaltensflexibilität beschreibt die Verfügbarkeit eines Verhaltensrepertoires von Aufgaben, auf dessen Basis eine Variante zur Gestaltungszeit (Buildtime) bestimmt, oder während der Ausführungszeit (Runtime) gewählt werden kann. Im Allgemeinen wird eine Aufgabe versuchen, interne oder externe Veränderungen im Rahmen ihres bestehenden Repertoires zu bewältigen. Ist ihr definiertes Verhaltensspektrum hierfür nicht mehr ausreichend, so sind Änderungen an der Aufgabenstruktur erforderlich ([BBF+11, S. 2], [WSL+11a, S. 90 f.]).

Strukturflexibilität entsteht durch „Anpassung“, also dem *Hinzufügen* oder *Entfernen* von Aufgabenmerkmalen. Eine Modifikation der Aufgabenstruktur verfolgt somit insbesondere das Ziel einer Anpassung des vorhandenen Verhaltensrepertoires, um auf interne oder externe Veränderungen besser reagieren zu können. Darüber hinaus dient Strukturflexibilität auch zur Effizienzverbesserung der Aufgabendurchführung bei unverändertem Verhalten ([BBF+11, S. 2 f.], [WSL+11a, S. 90 f.]).
Die Flexibilität betrieblicher Aufgaben fasst Tabelle 4.1 zusammen.

		Flexibilitätsarten	
		Verhaltensflexibilität	**Strukturflexibilität** Hinzufügen/Entfernen von…
Aufgabenmerkmale	**Vorereignis**	Variabler zeitlicher Eintritt von Vorereignis(sen)	Vorereignis(sen)
	Sach-/Formalziel	Variabilität von Sach-/ Formalziel(en)	Sach-/ Formalziel(en)
	Lösungsverfahren	Ergebnisvariabilität des Lösungsverfahrens	Bestandteilen eines Lösungsverfahrens
	Aufgabenobjekt	Veränderlichkeit von Attributen	Attribut(en)
	Nachereignis	Variabler zeitlicher Eintritt von Nachereignis(sen)	Nachereignis(sen)

Tabelle 4.1: Flexibilität betrieblicher Aufgaben [WSL+11a, S. 90]

Die Definition von Flexibilität auf der IS-Aufgabenebene ermöglicht zum einen die Übertragung dieses Verständnisses auf Geschäftsprozessmodelle, die den ereignisgesteuerten Ablauf von Aufgaben beschreiben. Auf Basis des Aufgabekonzepts ist zum anderen die Bereitstellung geeigneter methodischer Hilfsmittel möglich, um eine Überführung von Flexibilität der Aufgabenebene auf die Aufgabenträgerebene zu unterstützen.

4.2.3 Flexibilität in SOM-Geschäftsprozessmodellen

Geschäftsprozesse spezifizieren das Aufgabensystem der betrieblichen Leistungserstellung und -koordination [FeSi13, S. 189 f.]. Im Folgenden dient das Aufgabenkonzept als methodisches Fundament für das Übertragen des Flexibilitätsbegriffs der Aufgabe auf Geschäftsprozesse. Zu berücksichtigen sind dabei folgende Merkmale von Geschäftsprozessen [BBF+11, S. 3]:

- Aus verhaltensorientierter Sicht stellen Geschäftsprozesse einen ereignisgesteuerten Ablauf von Aufgaben dar, die sich im Hinblick auf die Erreichung gemeinsamer Ziele koordinieren (Abschnitt 4.1.2).
- Aus strukturorientierter Sicht wird die (Struktur von) Geschäftsprozessen als ein Netz betrieblicher Aufgaben (Aufgabennetz) beschrieben, dessen Verhalten durch die Interaktion zwischen diesen Aufgaben bestimmt wird ([WSL+11a, S. 88], Abschnitt 4.2.2).

Als Erweiterung des Flexibilitätsverständnisses von betrieblichen Aufgaben erstreckt sich Flexibilität in Geschäftsprozessen zudem auf die Aufnahme von externen Inputs, Durchführung der Leistungserstellung und deren Übergabe an die Umwelt (Output) sowie die Zuordnung und Nutzung von Ressourcen.

Das erarbeitete IS-Flexibilitätsverständnis wird im Folgenden auf SOM-Geschäftsprozessmodelle übertragen. Die **Flexibilität in SOM-Geschäftsprozessmodellen** ist durch drei Merkmale charakterisiert ([BBF+11, S. 3] und Abschnitt 4.2.1):

- *Zielorientierung*: Flexibilität von SOM-Geschäftsprozessmodellen dient deren Anpassung an Umwelt- oder Systemänderungen im Sinne der evolutionären Weiterentwicklung. Sie erfolgt stets im Hinblick auf die Erfüllung des übergeordneten Ziels, die Überlebensfähigkeit des betrieblichen Systems zu sichern.
- *Verhaltensflexibilität*: SOM-Geschäftsprozessmodelle besitzen ein definiertes Verhaltensrepertoire, mit dem sie zu einem gewissen Grad auf interne und externe Veränderungen mittels Auswahl von Verhaltensvarianten reagieren können.
- *Strukturflexibilität*: Das Anpassen der Struktur von SOM-Geschäftsprozessmodellen erfolgt durch das Hinzufügen oder Entfernen von Modellbausteinen und den Beziehungen zwischen diesen Bausteinen.

VERHALTENSFLEXIBILITÄT IN SOM-GESCHÄFTSPROZESSMODELLEN

Verhaltensflexibilität in SOM-Geschäftsprozessmodellen beschreibt eine *Variabilität im Ablauf des Geschäftsprozesses*. Durch seine Verhaltensflexibilität ist der Geschäftsprozess damit zu einem gewissen Grad gegenüber externen oder internen Veränderungen „robust“ (z. B. [AlSc95, S. 7]). Die Bestandteile des ereignisgesteuerten Aufgabennetzes, die im SOM-GPM zu variablem Verhalten führen, sind Aufgaben sowie die internen/externen Ereignisse (und Transaktionen) zwischen Aufgaben. Darüber hinaus spezifizieren die Aufgabenmerkmale *Ziele*, *Lösungsverfahren* sowie *Aufgabenobjekt* das Verhaltensrepertoire. Zur Gestaltungszeit oder während der Ausführungszeit des Geschäftsprozesses wird aus diesem Repertoire eine Variante ausgewählt. Die Wahl einer Variante kann sich wiederum auf die art-, mengen- und zeitmäßige Realisierung des Outputs eines Geschäftsprozesses auswirken [BBF+11, S. 2 f.]. Beispielsweise ist die Modellierung einer Menge von Ablaufvarianten in Form alternativer Nachereignisse im VES denkbar. Die Wahl und Ausführung einer Variante erfolgt auf Basis der vorgegebenen Ziele oder den Merkmalen eines eintretenden Vorereignisses ([BBF+11, S. 2 f.], [WSL+11a, S. 99 ff.]).

Reicht die Verhaltensflexibilität im SOM-Geschäftsprozessmodell nicht mehr aus, um auf interne oder externe Veränderungen zu reagieren, so ist die Anpassung seines Verhaltensspektrums notwendig. Dies wird durch die Modifikation der Modellstruktur realisiert.

STRUKTURFLEXIBILITÄT IN SOM-GESCHÄFTSPROZESSMODELLEN

Strukturflexibilität manifestiert sich in SOM-Geschäftsprozessmodellen durch die Anpassung von Modellbausteinen und Beziehungen zwischen den Bausteinen. Auf Basis der Metaobjekte des integrierten SOM-Metamodells der Geschäftsprozessebene (GP-Bausteine) und der Festlegung von Operationen zu deren Manipulation, lassen sich potenzielle Strukturänderungen auf der Aufgabenebene bestimmen [WSL+11a, S. 90 f.]. Die Struktur von SOM-Geschäftsprozessmodellen wird durch Manipulation der Bausteine *betriebliches Objekt*, *betriebliche Transaktion*, *Aufgabe*, *Leistung*, *objektinternes Ereignis*, *Umweltereignis* und *zeitkontinuierliche*

Parametrisierung angepasst. Dabei erfolgt die Änderung eines GP-Bausteins stets zusammen mit seinen Beziehungen (vgl. SOM-Metamodell in Abbildung 4.5).

Strukturelle Änderungen in einem SOM-Geschäftsprozessmodell sind anhand der **elementaren Änderungsoperationen**

- *Hinzufügen* und
- *Entfernen*

von Bausteinen sowie Beziehungen zwischen Bausteinen beschreibbar (z. B. [WRR07, S. 585]). Auf Basis dieser elementaren Operationen können *komplexe Änderungsoperationen* für die Manipulation der Struktur von SOM-Geschäftsprozessmodellen definiert werden, die dann als Folge elementarer Operationen spezifiziert sind (z. B. [WRR07, S. 579], [CDP07, S. 167 f.]). Als Beispiele seien „Verschieben“ oder „Ersetzen“ von einzelnen Bausteinen oder Teilen des Modells (einschließlich der relevanten Beziehungen) genannt. Eine Manipulation der Eigenschaften von GP-Bausteinen kann zudem durch die Operation „Anpassen“ oder „Aktualisieren“ erfolgen und nimmt damit Bezug auf das Verhalten des Geschäftsprozesses ([HRO+10, S. 5], [WRR07, S. 579], [CDP07, S. 170], [Allw98]). Zu berücksichtigen ist in diesem Zusammenhang, dass zwischen den Bausteinen in SOM-GPM existentielle Abhängigkeiten bestehen (vgl. SOM-Metamodell). Folglich zieht das Anpassen eines GP-Bausteins normalerweise weitere Modelländerungen nach sich.

In der vorliegenden Arbeit erfolgt die modellbasierte Spezifikation von RESTful SOA auf Basis strukturflexibler SOM-GPMs (Abschnitt 1.2.1). Aus der Perspektive des Systementwicklers stellt das Auftreten von strukturellen GP-Änderungen damit den Input für die Durchführung der Systementwicklungsaufgabe dar. Dagegen ist die Gestaltung von *robusten Geschäftsprozessen*, die eine möglichst hohe Verhaltensflexibilität aufweisen, oder *adaptiven Geschäftsprozessen*, die Strukturen und Mechanismen zur „selbstständigen“ Anpassung an Veränderungen besitzen, nicht Gegenstand der vorliegenden Arbeit. Für nähere Informationen zu diesen Themen sei auf die weiterführende Literatur verwiesen (z. B. [WKK+11], [WSR09], [Allw98], [Remm97], [AlSc95]).

4.3 Einführung der Fallstudie

Die Entwicklungsmethodik wird zur Erfüllung des Untersuchungsziels der vorliegenden Arbeit schrittweise in den nachfolgenden Kapiteln 5 und 6 konstruiert. Die Erläuterung und eine beispielhafte Anwendung der Methodik erfolgen dabei auf Basis einer Fallstudie aus dem Bereich der Abwicklung von Flugbuchungen. Nachfolgend wird die Fallstudie vorgestellt.

4.3.1 Untersuchungssituation

Die Untersuchungssituation der Fallstudie umfasst den Geschäftsprozess des *Direktvertriebs von Flugtickets*, der die Erbringung der Leistung *Verkauf von Flugtickets* durch eine *Fluglinie* an ihre *Kunden* beschreibt. Die Fallstudie wurde auf Basis von Beiträgen zum Flugbuchungsprozess ([KJL09], [BSL+08]) sowie der Airline IT Trend Survey 2013 [SITA13] erstellt.

Ein Verkauf bzw. eine Vermittlung von Flügen findet im **Geschäftsprozess** entweder direkt zwischen Fluglinie und Kunden, oder indirekt über einen autorisierten Agenten bzw. Vertriebspartner statt ([KJL09, S. 707 f.], [BSL+08, S. 8]). Bei der Durchführung einer direkten oder indirekten Flugvermittlung wird als Interaktionsarten grundlegend zwischen der

- „Offline"-Buchung von Tickets vor Ort, also z. B. im Reisebüro des Agenten oder am Ticketschalter der Fluglinie,
- telefonischer Buchung im Call-Center, sowie der
- „Online"-Buchung über eine (webbasierte) Anwendung

unterschieden [BSL+08, S. 14]. Den Abschluss der Buchungsleistung markiert die Übergabe der verkauften Flugtickets in elektronischer Form als *eTicket*, oder in physischer Form als Papierticket an den Kunden ([KJL09, S. 707 f.], [BSL+08, S. 14]).

Die Airline IT Trend Survey 2013 für internationale Fluglinien[24] zeigt eine starke Fokussierung auf den Ausbau von webbasierten und mobilen Anwendungen zur direkten Erbringung von Passagierdienstleitungen. Die Dienstleistungen *Flugsuche*, *Flugbuchung* oder auch der *Check-In* zwischen Fluglinie und Kunde werden bereits zu einem hohen Grad mittels internetbasierter Anwendungssysteme realisiert. Zudem wird für das Jahr 2016 prognostiziert, dass über 83% der Flugtickets in elektronischer Form den Passagieren zugestellt werden. Für die kommenden Jahre ist die Entwicklung von zusätzlichen Anwendungen für eine Vielzahl an Serviceleistungen, wie z. B. der Bereitstellung von Abfluginformationen für Passagiere, der Bearbeitung von Beschwerden (complaint handling) oder Umbuchungen (re-booking), geplant [SITA13, S. 8].

Eine **Abgrenzung** der Untersuchungssituation der Fallstudie wird nachfolgend vorgenommen:

- Der Geschäftsprozess der Fallstudie beschreibt die Abwicklung der direkten Buchung bzw. des direkten Verkaufs von Flügen zwischen Fluglinie und Kunde. Dabei wird eine vollständige Ausschöpfung des bestehenden Automatisierungspotenzials im Geschäftsprozess angestrebt. Für die weiteren Untersuchungen wird deshalb festgelegt, dass auch die Leistungsübergabe in elektronischer Form (eTicket) stattfindet.
- In der nahen Zukunft sind Anpassungen am Geschäftsprozess zu erwarten, der somit strukturflexibel ist. Beispielweise sind Erweiterungen um zusätzlich angebotene Dienstleistungen geplant. Für die Architektur des AwS wird deshalb gefordert, dass sie für eine flexible Weiterentwicklung geeignet ist.
- Die Durchführung des Geschäftsprozesses soll in der Fallstudie durch die Neu-Entwicklung eines modernen betrieblichen AwS unterstützt werden, welches die Automatisierungspotenziale der zu erbringenden Dienstleistungen zwischen Fluglinie und Kunden im Kontext des Direktvertriebs von Flügen realisiert. Die Leistung des Vertriebs

[24] Die *Airline IT Trends Survey 2013* der SITA wird im Auftrag der internationalen Luftfahrtindustrie unter den Top 200 Passagier-Fluglinien durchgeführt [SITA13, S. 14]. Als grundlegenden IT-Trend wird die Entwicklung von Anwendungssystemen zur Erbringung mobiler Passagierdienstleistungen identifiziert. So geben 97% der befragten Fluglinien an, in den Ausbau der IT-Infrastruktur zur Unterstützung dieser Vertriebskanäle und Dienstleistungen zu investieren.

von Flugtickets umfasst die *Flugsuche*, *Flugbuchung*, *Versand von eTickets* an den Kunden sowie die *Abrechnung*. Der Zugriff auf die Dienste des Systems soll internetbasiert über die Webseiten der Fluglinie (oder mittels eigener clientseitiger Anwendungen) erfolgen.

Die vorangegangenen Ausführungen beschreiben die Ziele und Rahmenbedingungen für die Durchführung der Systementwicklungsaufgabe der Fallstudie. Als Realisierungsform des Anwendungssystems wird die RESTful SOA gewählt. Die Zielarchitektur RESTful SOA bietet sich zur Lösung der aufgespannten Problemstellung an, da sie aufgrund des Merkmal der Ressourcenorientierung eine flexible Anpassung der Komponenten von RESTful SOA an die erwartete Strukturflexibilität des Geschäftsprozesses fördert. Zudem wird die Implementierung einer Vielzahl an Kommunikationskanälen mit dem Kunden auf Basis von HTTP unterstützt. Eine direkte MC-Kommunikation kann z. B. durch die Bereitstellung von HTML-Weboberflächen, oder die CC-Kommunikation durch den internetbasierten Zugriff von externen Anwendungen, z. B. mobilen Apps, erfolgen. Die Modellierung des Geschäftsprozesses erfolgt auf Basis der SOM-Methodik. Das SOM-Geschäftsprozessmodell der Fallstudie wird nun erstellt und schrittweise verfeinert.

4.3.2 SOM-Geschäftsprozessmodell

Das initiale IAS des SOM-GPM sowie die erste Detailstufe nach der Transaktionszerlegung zeigt Abbildung 4.10.

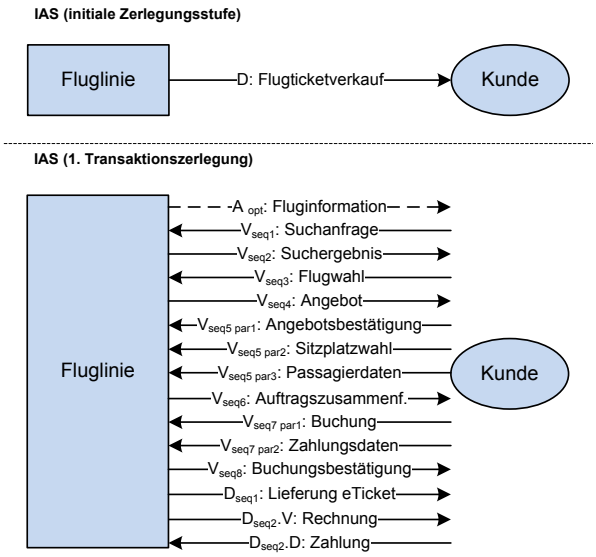


Abbildung 4.10: IAS der Fallstudie (initiale Stufe und erste Transaktionszerlegungsstufe)

Das initiale IAS besteht aus den betrieblichen Objekten *Fluglinie* (Diskursweltobjekt) und *Kunde* (Umweltobjekt). Als Leistung des Geschäftsprozesses wird der *Flugticketverkauf* an den Kunden erbracht. Zur Aufdeckung der Koordinationsbeziehungen zwischen den zwei betrieblichen Objekten wird die Transaktion *Flugticketverkauf* nach dem Verhandlungsprinzip zerlegt. Das Ergebnis ist in der ersten Zerlegungsstufe des IAS zusammengefasst. Auf eine Darstellung des korrespondierenden VES wird verzichtet, da der Kommunikationsablauf eine sequentielle Durchführung der Transaktionen vorsieht. In der Vereinbarungsphase, die aus insgesamt acht

Vereinbarungstransaktionen besteht, werden die Details der Leistungserstellung und -übergabe zwischen Kunde und Fluglinie festgelegt[25].

Die Anbahnung zum Kunden ist optional und erfolgt z. B. mittels Übersendung von *Flug-informationen*. Im ersten Schritt der Vereinbarung übermittelt ein Kunde seine *Suchanfrage* und spezifiziert darin Start-/Zielflughafen, das Datum von Hinflug oder Hin-/Rückflug sowie die Anzahl der Passagiere („einfache Suche“). Als *Suchergebnis* werden ihm von der Fluglinie eine Menge passender Flüge mit den Details zu genauen Flugzeiten und Preisen zurückgeliefert. Auf Basis der Ergebnisliste trifft der Kunde eine *Flugwahl* und erhält daraufhin das *Buchungs-angebot* zu einem bestimmten Flug mit dem finalen Ticketpreis. Sagt dem Kunden das Angebot zu, so wird dieses *bestätigt* und zusammen mit den *Passagierdaten* und der *Sitzplatzwahl* an die Fluglinie übermittelt. Daraufhin wird eine *Auftragszusammenfassung* des neuen Buchungs-auftrags erstellt und die Details dem Kunden in übersichtlicher Form dargestellt. Die kosten-pflichtige Buchung des Fluges erfolgt durch die *Bestätigung des Auftrags* und die Eingabe der *Zahlungsdaten* sowie die anschließende *Buchungsbestätigung*. Die *Durchführungsphase* be-steht aus einer Durchführungstransaktion, welche die Lieferung von *eTickets* an den Kunden beinhaltet. Die *Abrechnung* (Rechnung, Zahlung) erfolgt auf Basis der eingegebenen Zahlungs-daten.

Auf die Detaillierung der Kommunikation zwischen Diskurswelt und Umwelt, folgt die Zerle-gung des Diskursweltobjekts. Das Ergebnis der Objektzerlegung zeigt Abbildung 4.11 anhand von IAS und einem Ausschnitt des VES des finalen SOM-Geschäftsprozessmodells.

Die Fluglinie wird nach dem Verhandlungsprinzip in die betrieblichen Objekte *Flugvertrieb* und *Flugbetrieb* zerlegt, um das Lösungsverfahren der Leistungserstellung und -koordination weiter zu detaillieren. Der Flugbetrieb erbringt die Serviceleistung *Flugverwaltung* an den Flugvertrieb und umfasst die Aufgaben zur Verwaltung von Flugplänen sowie der freien und reservierten Sitzplätze von Flügen[26]. Während einer Durchführung des Geschäftsprozesses werden vom Flugbetrieb die *Status von Flügen* angefragt und *Reservierungen* getätigt. Nach dem Regelungs-prinzip erfolgt im nächsten Schritt die Zerlegung des Flugvertriebs in das Lenkungsobjekt *Vertrieb* und die Leistungsobjekte *Versand* und *Rechnungswesen*. Nach erfolgreicher Flugreser-vierung weist der Vertrieb den Versand mit der Lieferung von Flugtickets in elektronischer Form (eTickets) an und beauftragt das Rechnungswesen mit der *Buchungsabrechnung*. Mit der Leistungsübergabe ist die Durchführung des Geschäftsprozesses *Flugbuchung* abgeschlossen.

---

[25] Der Ablauf der Flugbuchung zwischen Kunde und Fluglinie entspricht dem Online-Buchungsprozess zur Passagierbeförderung bei den Fluggesellschaften *Air Berlin* sowie *Lufthansa* (Webseite: http://www.airberlin.com/booking/flight/vacancy.php und http://www.lufthansa.com/online/portal/lh/de/booking, letzter Abruf beider Webseiten am 2. März 2015). Aus Übersichtlichkeitsgründen wur-de die Sitzplatzwahl als Teil der Buchung aufgenommen und kein eigener Prozessschritt modelliert.

[26] Das betriebliche Objekt *Flugbetrieb* einer Fluggesellschaft greift wiederum auf ein globales Distribu-tionssystem (Global Distribution System, GDS) zu, das für eine Vielzahl von Fluggesellschaften die Reservierung von Flügen und die aktuellen Flugpläne zentral verwaltet [Fisc14]. Beispiel für ein verbreitetes GDS europäischer Fluggesellschaften ist *Amadeus* (http://www.amadeus.com/, Abruf am 2. März 2015). In der vorliegenden Fallstudie wird vom Zugriff auf das externe GDS abstrahiert. Dessen Realisierung ist im Objekt *Flugbetrieb* gekapselt.

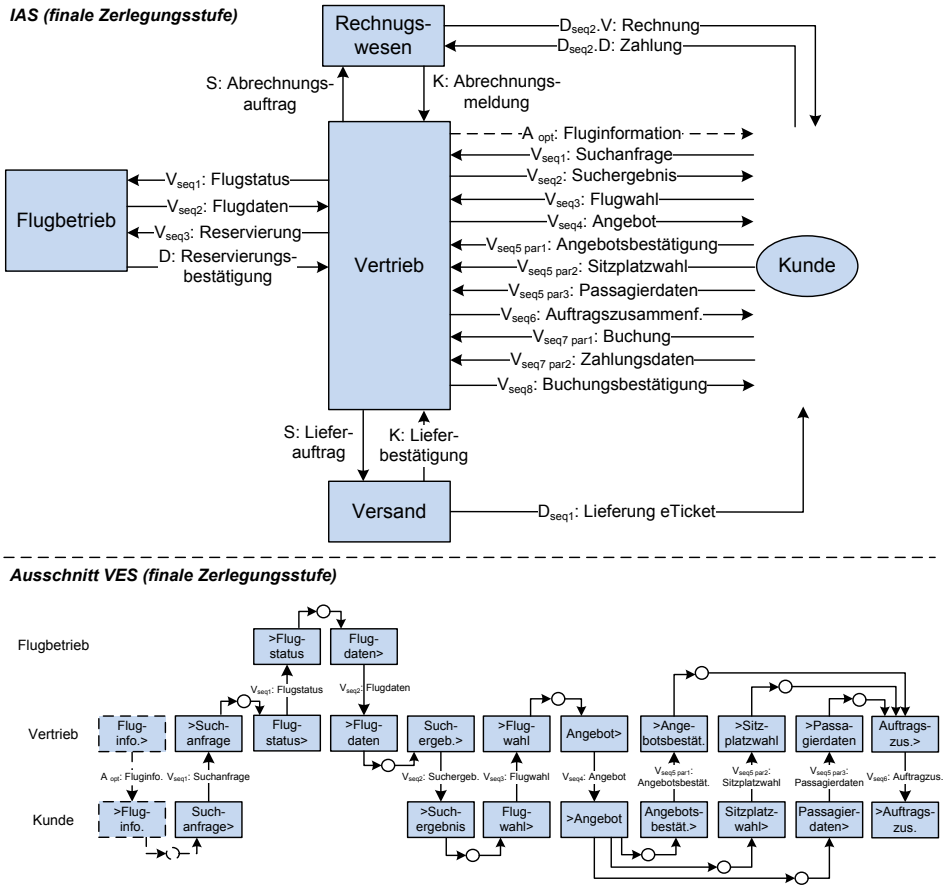


Abbildung 4.11: Finales IAS und Ausschnitt des finalen VES der Fallstudie

Das vollständige VES des SOM-Geschäftsprozessmodells der Fallstudie findet sich in Anhang A.

4.3.3 Automatisierung des Geschäftsprozesses

Für den Geschäftsprozess der Fallstudie wird angenommen, dass er vollständig von dem zu entwickelnden AwS durchgeführt werden soll. Allen Aufgaben und Transaktionen der Diskurswelt wird deshalb der Automatisierungsgrad *vollautomatisiert* zugewiesen. Dies ist möglich, da der Geschäftsprozess ausschließlich aus informationsverarbeitenden Aufgaben (Informationssystem) besteht und folglich kein Basissystem vorhanden ist. Der Einsatz personeller Aufgabenträger ist nicht vorgesehen.

Zur Koordination der Leistungsabwicklung kommunizieren das Diskursweltobjekt *Vertrieb* und das Umweltobjekt *Kunde* ausschließlich über die Schnittstellen des internetbasierten AwS. Die Transaktionen für die Koordination des Leistungsaustausches zwischen beiden Objekten werden somit in automatisierter Form übermittelt. Die Aufgaben des Vertriebes, die mit dem Empfang bzw. Senden der Transaktionen korrespondierenden, werden ebenfalls vollautomatisiert durchgeführt. Das Leistungsobjekt *Versand* erstellt und übergibt die *Flugtickets* auf elektronischem Wege an den *Kunden*. Die Aufgabe und die Transaktion der Leistungsübergabe laufen ebenfalls *vollautomatisiert* ab. Ein Versand von Flugtickets in Papierform ist bei der Online-Buchung nicht vorgesehen.

4.3.4 Abgrenzung des Anwendungssystems

Die Systementwicklung verfolgt in der eingeführten Fallstudie das Ziel der vollständigen Unterstützung des Geschäftsprozesses *Flugbuchung* durch die Neu-Entwickelung eines betrieblichen Anwendungssystems. Die Aufgaben und Transaktionen der betrieblichen Objekte *Vertrieb*, *Versand* und *Flugbetrieb* beschreiben somit das „Modell des betrieblichen Objektsystems“ und bilden den Ausgangspunkt für die Durchführung der Systementwicklungsaufgabe. Eine fachliche und softwaretechnische AwS-Spezifikation erfolgt im Rahmen der Konzeption der Entwicklungsmethodik (Kapitel 5).

4.4 Zusammenfassung

Die SOM-Methodik bildet in der vorliegenden Arbeit den methodischen Rahmen für die Modellierung von Geschäftsprozessen. Auf Basis des objekt- und geschäftsprozessorientierten SOM-Modellierungsansatzes erfolgt die struktur- und verhaltensorientierte Beschreibung von Geschäftsprozessen auf Aufgabenebene und die fachliche Spezifikation von Anwendungssystemen auf Aufgabenträgerebene. Die Bereitstellung integrierter Metamodelle unterstützt die Modellierung auf beiden Modellebenen sowie die Überführung von SOM-Geschäftsprozessmodellen in fachliche SOM-Anwendungsmodelle mittels metamodellbasierter Transformation. Insbesondere die explizite Spezifikation der strukturorientierten Sicht auf der Aufgaben- und Aufgabenträgerebene der SOM-Architektur ist dabei ein wichtiges Instrument für die Identifikation und Spezifikation von Bestandteilen der RESTful SOA in Kapitel 5.

Zur Sicherung der Überlebensfähigkeit betrieblicher Systeme sind deren Geschäftsprozesse flexibel, d. h. die Systeme passen ihre Struktur und/oder Verhalten an, um auf interne und externe Veränderungen zu reagieren oder diese zu antizipieren. Unter Verhaltensflexibilität wird in diesem Zusammenhang verstanden, dass bestehende Geschäftsprozesse ein definiertes Verhaltensrepertoire besitzen, mit dem sie durch die Auswahl einer Variante zu einem gewissen Grad auf Änderungen reagieren können. Reicht dieses Repertoire nicht mehr zur Bewältigung von Veränderungen aus, so ist normalerweise eine Verhaltensmodifikation durch Anpassung der Struktur des Geschäftsprozesses notwendig. In SOM-Geschäftsprozessmodellen spiegelt sich Strukturflexibilität im Hinzufügen und Entfernen von Bausteinen oder der Beziehungen zwischen Bausteinen wider. Eine Folge von Strukturflexibilität ist eine verringerte Abstimmung zwischen dem geänderten SOM-GPM und unterstützenden AwS. Aus der Flexibilität in Geschäftsprozessen entstehen Anforderungen an die Flexibilität der AwS zur Realisierung einer zielgerichteten Abstimmung zwischen Geschäftsprozess- und IT-Ebene.

Die vorangegangenen Ausführungen zur IS-Flexibilität stützen die Charakterisierung von RESTful SOA als *flexibler Systemarchitektur*. Die Bereitstellung von lose gekoppelten Ressourcen mit einheitlicher Schnittstelle und die Nutzung des standardisierten Anwendungsprotokolls HTTP adressieren die Merkmale *Kompatibilität* und *Modularität* der Systemkomponenten, und folglich die Anpassbarkeit des AwS. Der Einsatz von URIs und der Aufbau der Hypermedia zielen zudem auf die *Verknüpfbarkeit* von Komponenten und somit die verteilte Nutzbarkeit von Ressourcen. RESTful SOA weisen demnach zentrale Eigenschaften flexibler Architekturen auf.

5 Modellbasierte Gestaltung von Informationssystemen mit der SOM-R-Methodik

Die Entwicklungsmethodik zur modellbasierten Gestaltung und Pflege von RESTful SOA auf Basis flexibler SOM-Geschäftsprozessmodelle wird in den Kapiteln fünf und sechs dieser Arbeit konstruiert. Die nachfolgenden Ausführungen bauen auf den eingeführten Grundlagen der modellbasierten Systementwicklung, der RESTful SOA und dem Verständnis von flexiblen SOM-Geschäftsprozessmodellen auf.
Gegenstand des fünften Kapitels ist die Erweiterung der SOM-Methodik zur modellbasierten Gestaltung von RESTful SOA auf Basis von SOM-Geschäftsprozessmodellen. Das Kapitel beschreibt demnach den Lösungsansatz zur Erreichung des ersten Untersuchungsziels der Arbeit. Ausgehend von einer Bestimmung der Anforderungen an die Erweiterung der SOM-Methodik werden Architekturmodell, REST-Modellierungsansatz, Vorgehensmodell sowie Entwicklungsschritte der Methodik dieser Arbeit schrittweise konzipiert. Anhand der Fallstudie wird die erarbeitete Entwicklungsmethodik erläutert und ihre Anwendbarkeit gezeigt. Abschließend erfolgt eine zusammenfassende Betrachtung der Ergebnisse.

5.1 Anforderungen an die Konstruktion der SOM-R-Methodik

Im Folgenden werden die konzeptuellen Anforderungen an eine Entwicklungsmethodik als Lösungsansatz zur Erfüllung des ersten Untersuchungsziels dieser Arbeit betrachtet. Die allgemeinen *Bestandteile von Entwicklungsmethodiken* dienen als Startpunkt für die Bestimmung von Anforderungen an die vorliegende Konstruktionsaufgabe (Abschnitt 2.2.5). Die Methodik bzw. ihre Bestandteile werden dabei als „RESTful Erweiterung“ der SOM-Methodik von FERSTL und SINZ [FeSi13, S. 194 f.] entworfen. Die Methodik der vorliegenden Arbeit wird als *SOM-R-Methodik* bezeichnet. Die Konstruktion von Architekturmodell und Entwicklungsvorgehen der SOM-R-Methodik des fünften Kapitels ist Voraussetzung dafür, dass im sechsten Kapitel eine Erweiterung der Methodik um Konzepte zur Anpassung von RESTful SOA an Änderungen in flexiblen SOM-Geschäftsprozessmodellen durchgeführt werden kann.

Die Betrachtung der zu konstruierenden Artefakte *Architektur-* und *Vorgehensmodell* erfolgt in Abschnitt 5.1.1. Die modellbasierte Spezifikation von RESTful SOA aus SOM-GPMs erfordert die Erweiterung von Architektur- und Vorgehensmodell der SOM-Methodik um softwaretechnische Modellebenen. Die Anforderungen bezüglich der Konzeption eines *Modellierungsansatzes* zur Unterstützung der modellbasierten Gestaltung von Bestandteilen und Aufbau des Zielsystems *RESTful SOA* werden in Abschnitt 5.1.2 bestimmt.

Der durchzuführenden Konstruktionsaufgabe werden Formalziele an die Seite gestellt (Abschnitt 1.2.1). Die Methodik soll die Systementwicklung bei der Komplexitätsbewältigung der modellbasierten Entwicklung von RESTful SOA und deren konsistenter Abstimmung mit den fachlichen Anforderungen des SOM-GPMs unterstützen. Des Weiteren wird die flexible Anpassbarkeit der Methodik an geänderte (technologische) Rahmenbedingungen gefordert.

5.1.1 Erweiterung der SOM-Methodik

Eine Entwicklungsmethodik umfasst einen Modellierungsansatz, ein Architekturmodell, ein Vorgehensmodell und ggf. ein Softwarewerkzeug [Sinz13c, Kapitel 1, S. 9]. Im Folgenden werden die zu konstruierenden Bestandteile *Architekturmodell* und *Entwicklungsvorgehen* der SOM-R-Methodik bestimmt und auf dieser Basis die Anforderungen an die Erweiterung der SOM-Methodik herausgearbeitet. Den konzeptuellen Ausgangspunkt der weiteren Ausführungen bilden der Modellierungsansatz, das Architekturmodell und das Vorgehensmodell zur Beschreibung der Ebenen *betriebliches Objektsystem* und *Anwendungsmodell* der SOM-Methodik (Abschnitt 4.1).

Die modellbasierte Spezifikation des betrieblichen Anwendungssystems RESTful SOA setzt die Erweiterung der SOM-Methodik um softwaretechnische Beschreibungsebenen voraus. Ausgehend von den Beschreibungsebenen der Systementwicklung sind dies die **Modellebenen Softwarearchitektur** und **Implementierung** (Abschnitt 2.2.2). Beide Modellebenen sind in die bestehende Architektur der SOM-Methodik derart zu integrieren, dass die plattformspezifische Implementierung von RESTful SOA aus SOM-Geschäftsprozessmodellen in einem durchgängig modellbasierten Prozess entwickelt werden kann.

Die Erweiterung des SOM-Architekturmodells um neue Modellebenen setzt die Konzeption der Bestandteile *integriertes Metamodell*, *Sichten* und ggf. *Pattern* (Strukturmuster) für jede Modellebene voraus. Zudem ist eine Einordnung der neuen Modellebenen in die bestehende Modellebenenhierarchie sowie die Definition von Beziehungen zwischen benachbarten Modellebenen notwendig. Jede Modellebenen-Beziehung umfasst ein *Beziehungs-Metamodell* und ggf. *Beziehungs-Pattern*, welche ebenfalls zu gestalten sind (Abschnitt 2.1.6).

Die **Modellbildung** der Ebenen Softwarearchitektur und Implementierung erfolgt in der SOM-R-Methodik nach dem Schema des Entwicklungsvorgehens der SOM-Methodik. Dieses sieht die Trennung der Entwicklungsschritte in (automatisierbare) *Transformationen* und (nicht-automatisierbare) *Konsolidierungen*[27] (Überarbeitungen) vor (Abschnitt 4.1.1). Das Modellsystem wird dabei als Folge von Transformations- und Konsolidierungsschritten erstellt. Die Modellbildung in der SOM-R-Methodik erfordert somit:

- Die *Definition der Konsolidierungsschritte* zur Spezifikation der Modellebenen Softwarearchitektur und Implementierung der RESTful SOA.
- Die *Prüfung und ggf. Anpassung* der *Konsolidierungsschritte* für die Modellebenen SOM-Geschäftsprozessmodell und fachliche AwS-Spezifikation im Hinblick auf die Anforderungen der Zielarchitektur.
- Die *Definition der Transformationen* zur Ableitung von Softwarearchitektur und Implementierung.

[27] Der Begriff *Konsolidierung* wird im Sinne einer Vertiefung des Abstraktionsgrades einer Modellebenen-Beschreibung verstanden und meint damit sowohl die strukturelle Überarbeitung des Modellsystems, als auch die Anreicherung von dessen Bestandteilen um Ebenen-spezifische Details. Im Zusammenhang mit der Modellbildung auf den Ebenen eines Architekturmodells werden die Begriffe *Konsolidierung* sowie *Überarbeitung* und *Anreicherung* deshalb *synonym* verwendet.

Im weiteren Verlauf werden der Architekturrahmen der SOM-R-Methodik als **SOM-R-Architekturmodell** und die Modellebene der „Softwarearchitektur der RESTful SOA“ als *REST-Architekturmodell* bezeichnet. Den schematischen Aufbau des SOM-R-Architekturmodells sowie die Abfolge der Entwicklungsschritte als grundlegendes Vorgehen der Modellerstellung veranschaulicht Abbildung 5.1.

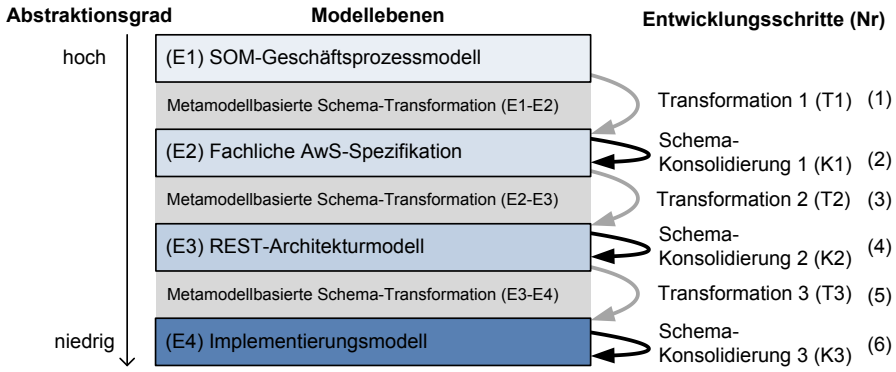


Abbildung 5.1: Schematischer Aufbau des SOM-R-Architekturmodells

Die Transformationen zur Überführung der Schemata benachbarter Ebenen werden in Form von *metamodellbasierten Transformationen* (T) realisiert. Metamodellbasierte Transformationen setzen die Metaobjekte zweier benachbarter Ebenen zueinander in Beziehung und ermöglichen damit eine automatisierte Überführung von deren Schemata. Ein derart abgeleitetes Schema liegt in initialer Form vor. Die Überarbeitung der initial abgeleiteten Modelle und die Anreicherung ihrer Bausteine um spezifische Merkmale der jeweiligen Modellebene erfolgt in einem *Konsolidierungsschritt* (K). Die *Konsolidierungsschritte* (Überarbeitungsschritte) der Methodik dienen der Unterstützung des Modellierers bei der semantischen Anreicherung der Ebenen des zu bildenden Modellsystems und kapseln hierzu das Wissen zur Realisierung nicht-automatisierter Entwurfsentscheidungen. Eine überarbeitete (konsolidierte) Modellebene ist hinreichend detailliert, um die Transformation in die nächsttiefere Ebene durchzuführen.

Ausgehend vom SOM-GPM werden die Ebenen fachliche AwS-Spezifikation, REST-Architekturmodell und Implementierungsmodell jeweils in initialer Form abgeleitet (Transformationen T1-T3) und anschließend konsolidiert (Konsolidierungen K1-K3). Die Abfolge der Transformations- und Konsolidierungsschritte zur Spezifikation des Modellsystems legt das **SOM-R-Vorgehensmodell** fest.

Die Erweiterung der SOM-Methodik um eine softwaretechnische Modellebene setzt zudem die Bestimmung eines Meta-Metamodells voraus. In dieser Arbeit wird das **Meta-Metamodell** nach SINZ [Sinz96] genutzt, da es bereits der Modellierung in der SOM-Methodik zugrunde liegt (Abschnitt 2.1.5). Der Einsatz einer gemeinsamen Metamodellierungsbasis fördert die Bereitstellung eines einheitlichen Begriffssystems im Modellsystem und somit die (modellbasierte) Überführung von Bausteinen der SOM-GPM-Ebene auf die Ebene des Zielsystems RESTful SOA.

5.1.2 Spezifikation der REST-Zielarchitektur

Voraussetzung für die Spezifikation des *REST-Architekturmodells* ist die Bestimmung der zu modellierenden Bestandteile und Merkmale von RESTful SOA. Im Folgenden werden Anfor-

derungen an die Konzeption des REST-Modellierungsansatzes in dieser Arbeit herausgearbeitet. Die Basis hierfür bildet die vorgestellte softwaretechnische Architektur von RESTful SOA, deren schematischer Aufbau in Abbildung 5.2 gezeigt wird (vgl. Abbildung 3.7).

Durch den Zugriff auf die Ressourcen wird ihre Funktionalität von RESTful SOA genutzt. Die *Nutzerebene* beschreibt hierfür Schnittstellen-Objekttypen zur Realisierung der Kommunikation mit einem Client (Nutzer). Die Identifikation und Spezifikation von *Ressourcen* und damit ihrer *RESTful Services* bildet den Kern des Entwicklungsprozesses von RESTful SOA. Es wird hierbei zwischen *Entitäts-* und *Vorgangsdiensten* bzw. ihren *Entitäts-* und *Vorgangsressourcen* der SOA unterschieden (Abschnitt 3.3.3). Diese Bestandteile beschreiben die *Außensicht* des Anwendungssystems.

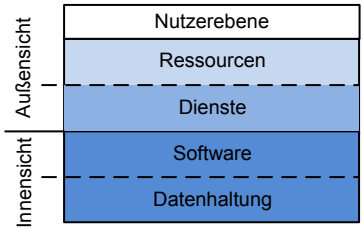


Abbildung 5.2: Schematischer Aufbau der RESTful SOA

Die primäre Quelle für die Identifikation von Ressourcen und der Festlegung ihres Ressourcentyps sind in dieser Arbeit fachliche Anforderungen. Die softwaretechnische Beschreibung von Ressourcen erfolgt durch die Spezifikation ihrer REST-Eigenschaften. Für jede zu entwickelnde Ressource ist die Bestimmung ihres einheitlichen Identifikators (URI), der standardisierten REST-Schnittstelle (HTTP), den Repräsentationen und Verknüpfungen zur Realisierung einer Hypermedia, sowie dem Ausnahme- und Antwortverhalten, einschließlich Response Codes und Medientypen, zur Realisierung der Interaktion mit Ressourcen notwendig. Des Weiteren sind Einstiegspunkte für das Anwendungssystem festzulegen (Abschnitt 3.3.2 sowie [ECP+13, S. 68 ff.], [Tilk11, S. 11 ff.], [RiRu07, S. 108 ff.]).

Aus der *Innensicht* wird die RESTful SOA durch *Softwarekomponenten* beschrieben, die wiederum eine Menge von Objekttypen beinhalten. Ressourcen verbergen ihre Realisierung durch Objekttypen hinter ihrer öffentlichen Schnittstelle. Die Entwicklung von Objekttypen erfordert insbesondere die Bestimmung und Spezifikation ihrer Attribute und (internen) Methoden sowie den Beziehungen zwischen Objekttypen. Die Persistierung von Zuständen des Anwendungssystems setzt zudem die Definition der Struktur der *Datenhaltung* voraus.
Ergänzend zu den geforderten Elementen eines REST-Modellierungsansatzes ist bei seiner Konzeption zudem darauf zu achten, dass er die Erstellung von REST-Architekturmodellen mit einer hinreichenden Spezifikationstiefe erlaubt, da diese den Ausgangspunkt für die modellgetriebene Ableitung der Implementierung von RESTful SOA bilden.

Im weiteren Verlauf des fünften Kapitels werden zunächst das Architekturmodell der SOM-R-Methodik (Abschnitt 5.2) sowie ein Ansatz zur Modellierung der Ebene des REST-Architekturmodells (Abschnitt 5.3) konzipiert. Anschließend werden das Vorgehensmodell (Abschnitt 5.4) sowie die Modelltransformationen und -konsolidierungen (Abschnitt 5.5) der Methodik schrittweise eingeführt. Die Zusammenfassung der Ergebnisse des Kapitels erfolgt in Abschnitt 5.6.

5.2 Konzeption des SOM-R-Architekturmodells

Die SOM-Unternehmensarchitektur bildet die Basis für die Konzeption des SOM-R-Architekturmodells. Diese wird um die Modellebenen *REST-Architektur* und *Implementierung* erweitert und geht demnach konform zu den vier klassischen Ebenen der Softwareentwicklung (Abschnitt 2.2.2). Der Unternehmensplan der SOM-Methodik ist nicht Gegenstand dieser Arbeit und wird nicht betrachtet (vgl. Abschnitte 1.2.1 und 4.1.1).

AUFBAU DES SOM-R-ARCHITEKTURMODELLS

Das erweiterte Architekturmodell der SOM-R-Methodik besteht aus den vier Modellebenen *SOM-Geschäftsprozessmodell* (E1), *fachliche AwS-Spezifikation* (fachliches Anwendungsmodell) (E2), *REST-Architekturmodell* (E3) und *Implementierungsmodell* (E4). Das Modellsystem jeder Ebene wird aus einer *strukturorientierten Sicht* und einer *verhaltensorientierten Sicht* betrachtet. Die Beziehungen zwischen benachbarten Modellebenen bauen die *Hierarchie* der Architektur auf. Sie sind als Beziehungs-Metamodelle (BMM) definiert. Als Extension des generischen Architekturrahmens unterstützen Metamodelle, Sichten und Strukturmuster (Pattern) die Modellierung der Ebenen des Modellsystems. Das **SOM-R-Architekturmodell** zeigt Abbildung 5.3.

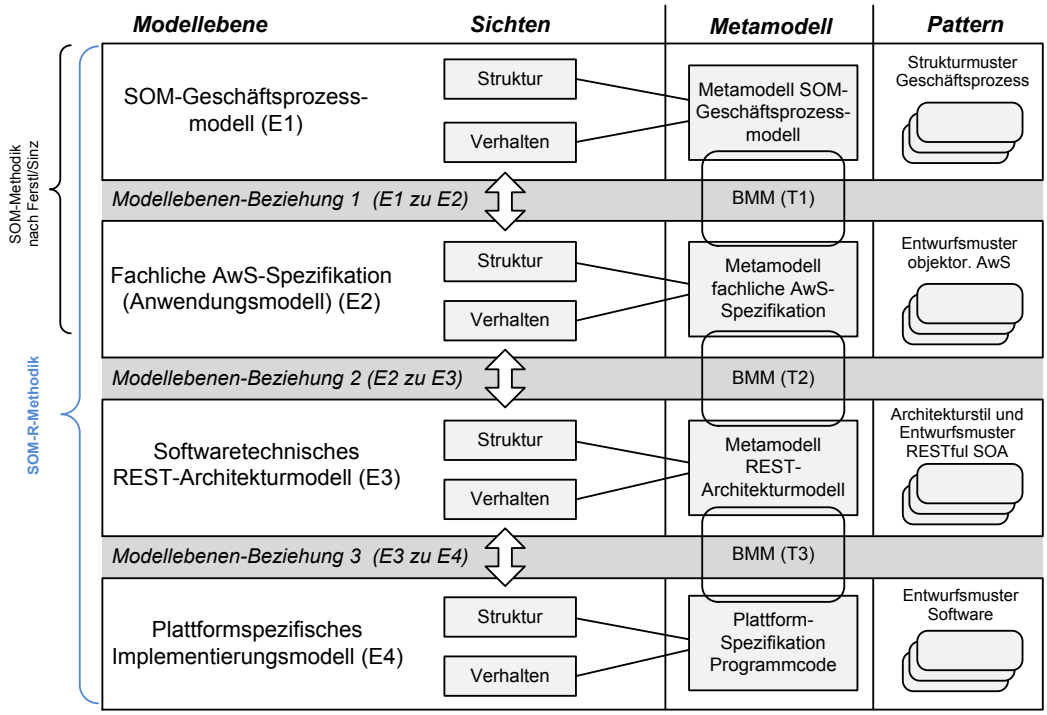


Abbildung 5.3: Architekturmodell der SOM-R-Entwicklungsmethodik

Die Metamodelle und Sichten von SOM-Geschäftsprozessmodell (E1) und fachlicher AwS-Spezifikation (E2) sind Bestandteile der SOM-Methodik von FERSTL und SINZ [FeSi13, S. 194 ff.] und werden in den Abschnitten 4.1.2 und 4.1.3 detailliert vorgestellt. Die Konstruktion des Metamodells der REST-Architekturmodellebene erfolgt in Abschnitt 5.3. Für die plattformspezifische Implementierung wird ein Metamodell in Abschnitt 5.5.8 exemplarisch gestaltet.

Die Anwendung elementarer (z. B. Regelungs- und Verhandlungsprinzip, Abschnitt 4.1.2) sowie domänenspezifischer Strukturmuster bietet sich für die Erstellung von SOM-Geschäftsprozess-

modellen an ([HSW98], [Sinz97]). Die fachliche AwS-Spezifikation ist durch Muster des objektorientierten Software-Entwurfs unterstützbar (z. B. [GHJ+94]). Im REST-Architekturmodell wird die Modellbildung grundsätzlich durch die Prinzipien des Architekturstils REST geleitet. Bei der Gestaltung von RESTful SOA sowie bei der Spezifikation ihrer Dienste und Ressourcen kann auf REST-spezifische Entwurfsmuster oder auch allgemeine Architekturmuster von SOA zurückgegriffen werden (z. B. [Alla10], [ECP+13, S. 327 ff.]). Auf der Ebene plattformspezifische Implementierung empfiehlt sich für objektorientierte Basismaschinen der Einsatz von Entwurfsmustern zur Realisierung objektorientierter Softwareprogramme (z. B. [GHJ+94], [Busc00]).

Sichten des SOM-R-Architekturmodells

Im Folgenden werden das **SOM-R-Modellsystem** und die auf seinen vier Modellebenen gebildeten **Sichten** näher betrachtet. Die modellierten Schemata der jeweiligen Sicht sowie die gewählte Repräsentationsform zeigt Abbildung 5.4. Ihre grundlegende Form wurde erstmals von Wolf [Wolf12, S. 1652] vorgestellt und in der Arbeit von Wolf und Benker [WoBe13, S. 1233] verfeinert.

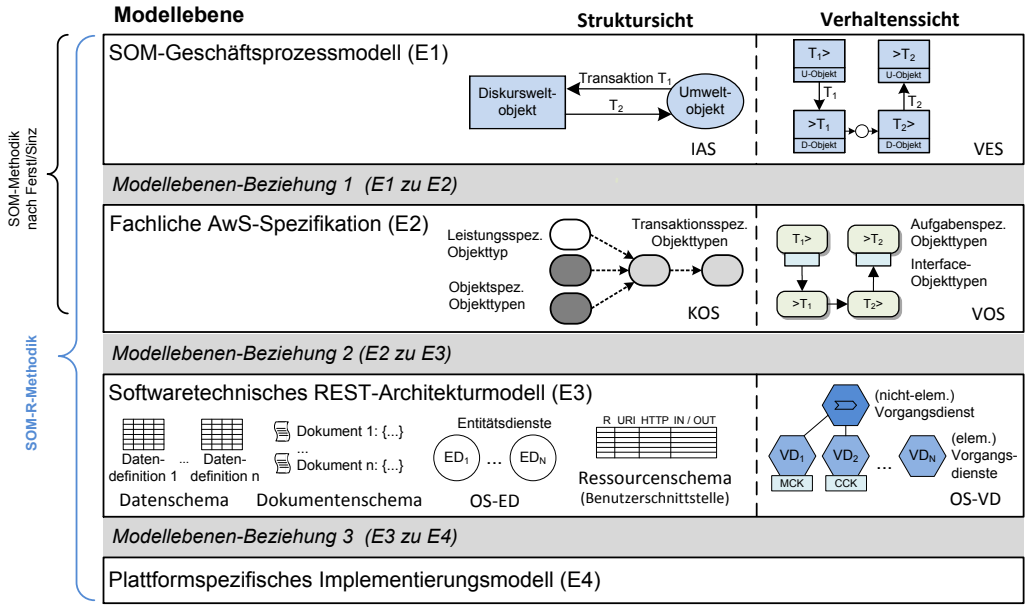


Abbildung 5.4: Sichtenbildung im SOM-R-Architekturmodell (in Anlehnung an [WoBe13])

Die Modellierung des *SOM-Geschäftsprozessmodells* erfolgt aus strukturorientierter Sicht im IAS und aus verhaltensorientierter Sicht im korrespondierenden VES. Das plattformunabhängige, objektorientierte *Modell des Anwendungssystems* wird ebenfalls aus der Strukturansicht im KOS und aus der Verhaltenssicht im VOS spezifiziert [FeSi13, S. 198 f.].

Die *fachliche AwS-Spezifikation* wird aus dem SOM-Geschäftsprozessmodell durch eine metamodellbasierte Transformation abgeleitet. Hierfür werden die Bausteine des SOM-Metamodells der Quellsprache (SOM-Geschäftsprozessmodell) mit den Metamodell-Bausteinen der Zielsprache (AwS-Spezifikation) in einer Abbildungsspezifikation verknüpft, was zur Definition der ersten Modellebenen-Beziehung führt [FeSi13, S. 235 f.]. Die metamodellbasierte Transformation wird in Abschnitt 5.5.2 dieser Arbeit erläutert.

Die softwaretechnische REST-Architektur (dritte Modellebene) erweitert die SOM-Architektur um eine plattformspezifische Aufgabenträgerebene zur Spezifikation der softwaretechnischen Architektur der RESTful SOA. Das *REST-Architekturmodell* beschreibt die Vorgangs- und Entitätsservices, die Schnittstellenspezifikation (Ressourcenschema) sowie die Schemadefinitionen der persistenten Datenhaltung und ausgetauschter Dokumente (vgl. Abschnitt 5.1.2).

Den Kern der RESTful SOA bilden die Objektschemata der Dienste. Das *Objektschema der Entitätsdienste* (OS-ED) spezifiziert die Realisierung der Ressourcen eines Entitätsdienstes jeweils in Form von Objekttypen (Struktursicht). Korrespondierend dazu werden die Objekttypen zur Realisierung der (elementaren und nicht-elementaren) Vorgangsressourcen sowie der Interface-Objekttypen im *Objektschema der Vorgangsdienste*[28] (OS-VD) beschrieben. Beide Objektschemata erfassen somit den Übergang zwischen Innen- und Außensicht der RESTful SOA.

Die Verwaltung und Persistierung von Ressourcenzuständen definieren das *Daten*- und das *Dokumentenschema*. Die Kommunikation von Clients mit RESTful Services erfolgt durch den Austausch von Zustandsbeschreibungen ihrer Ressourcen (Repräsentation) in Form von Dokumenten. Persistente Ressourcenzustände werden in der Datenhaltung verwaltet. Die Nutzung der Service-Funktionalität erfolgt über die veröffentlichte einheitliche Ressourcen-Schnittstelle, welche im *Ressourcenschema* beschrieben wird. Das Ressourcenschema spezifiziert demnach den Dienstvertrag einer RESTful SOA.

Das REST-Architekturmodell wird aus dem Anwendungsmodell abgeleitet und bildet den Input (Quellmodell) für die Durchführung der Transformation in das Modell der Implementierung (E4). Beide *Modellebenen-Beziehungen* (E2-E3, E3-E4) werden in Form *metamodellbasierter Transformationen* realisiert. Eine Untersuchung erfolgt in den Abschnitten 5.5.4 und 5.5.6.

Die plattformspezifische Implementierung des spezifizierten Anwendungssystems beschreibt das *Implementierungsmodell*, welches damit den Programmcode des entwickelten AwS für konkrete Basismaschinen erfasst. Die Spezifikation der Implementierungsebene setzt stets die Wahl geeigneter Basismaschinen voraus. Die Ableitung und Gestaltung dieser vierten Modellebene wird im Rahmen der Fallstudie gezeigt (Abschnitt 5.5.8).

5.3 Modellierungsansatz des REST-Architekturmodells

Die Konzeption des *Modellierungsansatzes* zur Erstellung des REST-Architekturmodells ist Gegenstand dieses Abschnitts. Die Anforderungen an die Modellierung von RESTful SOA bzw. ihrer Bestandteile wurden in Abschnitt 5.1 vorgestellt. Ergänzend dazu ergeben sich aus den Ausführungen zum SOM-R-Architekturmodell weitere Anforderungen an die Gestaltung der REST-Modellebene (Abschnitt 5.2).

Die Konzeption des Modellierungsansatzes beginnt zunächst mit der Einführung der *Metapher* (Abschnitt 5.3.1) und der Erläuterung der *Beziehungen zwischen zentralen Bausteinen* der

[28] Korrespondierend zu den VOS der fachlichen AwS-Spezifikation werden im REST-Architekturmodell i. d. R. mehrere OS-VD(s) spezifiziert. Diese werden in den nachfolgenden Erläuterungen gemeinsam adressiert und dabei in der Einzahl unter dem Begriff *OS-VD* zusammengefasst.

Modellebene (Abschnitt 5.3.2). Anschließend werden die *Metamodelle* zur Modellierung der verschiedenen Schemata des REST-Architekturmodells erläutert (Abschnitte 5.3.3–5.3.5).

5.3.1 Metapher

Die Metapher des REST-Architekturmodells baut auf der übergeordneten *Metapher der SOM-Methodik* auf [FeSi13, S. 197] und erweitert diese um die Charakteristika der *service-orientierten Architektur* und der *Ressourcenorientierung* (Abschnitte 3.1.2 und 3.3.2).

Die Modellierung des REST-Architekturmodells folgt der Metapher eines *„verteilten Systems, welches aus lose gekoppelten Diensten besteht, die ihre Funktionalität als Menge von Ressourcen mit einheitlicher Schnittstelle bereitstellen. Zur Erreichung bestimmter Ziele koordinieren sich die Dienste durch den Austausch von Nachrichten“* (eigene Definition).

5.3.2 Beziehungen zwischen zentralen Bausteinen der REST-Architektur

Die Beziehungen zwischen den Kernbausteinen der Schemata *OS-ED*, *OS-VD*, *Ressourcenschema* sowie dem *Daten-* und *Dokumentenschema* werden in einem vereinfachten integrierten Metamodell der REST-Architekturebene definiert. Auf die Visualisierung des detaillierten, integrierten REST-Metamodells wird verzichtet, da dieses eine hohe Komplexität aufweisen und damit die Erklärbarkeit erschweren würde. Stattdessen werden in den nachfolgenden Abschnitten 5.3.3 bis 5.3.6 die Metamodelle der gebildeten Sichten der Modellebene bzw. der hierdurch definierten Schemata vorgestellt. Eine Integration dieser (Teil-)Metamodelle erfolgt über die Beziehungen zwischen den zentralen Metaobjekten des REST-Architekturmodells, welche Abbildung 5.5 visualisiert.

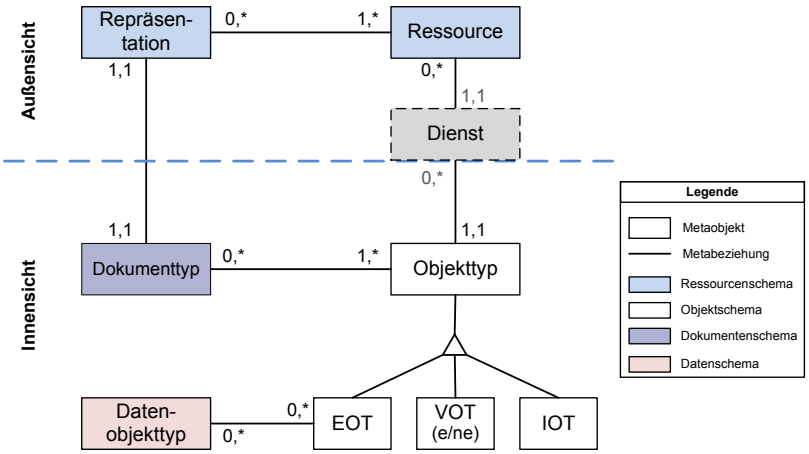


Abbildung 5.5: Integriertes Metamodell der REST-Architekturmodellebene (vereinfachte Darstellung zentraler Bausteine)

Aus der *Außensicht* veröffentlicht die RESTful SOA die Funktionalität ihrer Dienste in Form einer Menge von *Ressourcen*. Die Kommunikation zwischen einem Nutzer und einer Ressource erfolgt über den Austausch von *Repräsentationen*, die in Form von Nachrichten gesendet bzw. empfangen werden. Eine Ressource kann mehrere Repräsentationen bereitstellen. Jede Repräsentation spezifiziert einen Zustand, der eine oder mehrere Ressourcen beschreibt (Abschnitt 3.3.2.2).

Objekttypen, *Datenobjekttypen* und *Dokumenttypen* sind die zentralen Bausteine, welche die Realisierung der RESTful SOA aus der *Innensicht* spezifizieren. Den Übergang zwischen Außen- und Innensicht markiert das Konzept des *Dienstes* (Abschnitt 3.3.3.2). Der Begriff des Dienstes dient im REST-Architekturmodell als „fachliche Klammer“ und markiert den Übergang zur Innensicht (graue Füllung, gestrichelte Linie). Die Ressourcen eines Dienstes werden dabei durch genau einen Objekttypen realisiert. Objekttypen werden in Entitätsobjekttypen (EOT), elementare und nicht-elementare Vorgangsobjekttypen (eVOT und neVOT) sowie Interfaceobjekttypen (IOT) unterschieden (vgl. Serviceklassen in Abschnitt 3.3.3.1). Als *Modellierungseinschränkung* des REST-Architekturmodells wird in dieser Arbeit festgelegt, dass die Realisierung jedes Dienstes durch einen Objekttypen erfolgt. Der Dienstbegriff dient somit als Strukturierungsmerkmal im REST-Architekturmodell. Diese Festlegung der Beziehung zwischen Ressource, Dienst und Objekttyp wird jedoch auf der Implementierungsebene aufgehoben.

Die Verwaltung von persistenten Zuständen des Anwendungssystems führen EOTs durch, die hierzu mehrere Datenobjekttypen (DOT) der Datenhaltung manipulieren können. Transiente und persistente (Zustands-)Informationen werden in Form von Dokumenten ausgetauscht. Die Struktur des Dokuments einer Ressourcen-Repräsentation definiert ein Dokumenttyp (DT).

Die Modellierung elementarer und nicht-elementarer VOTs sowie IOTs erfolgt im OS-VD. Das OS-ED spezifiziert die EOTs. Darauf aufbauend wird im Ressourcenschema die öffentliche Schnittstellenbeschreibung der bereitgestellten Ressourcen von RESTful Services spezifiziert. Das Dokumentenschema definiert die Struktur ausgetauschter Repräsentationen in Form von DTs. Die Schemadefinition der Datenstruktur umfasst die DOTs und deren Beziehungen.
Während die Modellierung der *Datensicht* in Daten- und Dokumentenschema erfolgt, beschreiben OS-ED und OS-VD das betrachtete Anwendungssystem aus *Funktions-, Interaktions-* sowie *Vorgangssicht*. Das OS-ED markiert die Schnittstelle zwischen Datenhaltungsebene und Anwendungsebene (vgl. Ausführungen zu Abbildung 4.8).

Die vorgestellten Metaobjekte definieren die zentralen Elemente für das Begriffssystem des Modellierungsansatzes der REST-Architektur. Sie dienen im weiteren Verlauf der Verknüpfung zwischen den Kernbausteinen der RESTful SOA und des SOM-GPM und sind damit wichtige Voraussetzung für eine konsistente Abstimmung der Modellebenen im SOM-R-Architekturmodell.

5.3.3 Metamodell der RESTful Services

Die Dienste der RESTful SOA werden im REST-Architekturmodell aus der Innensicht durch die Schemata OS-ED und OS-VD, und aus der Außensicht durch das Ressourcenschema beschrieben. Im Folgenden wird das Metamodell zur Spezifikation von RESTful Services konzipiert. In dessen Fokus steht die Beschreibung der Innensicht von RESTful SOA. Ferner wird die Beziehung zur Außensicht des AwS auf Basis des Konzepts der Ressourcenorientierung hergestellt. Das *Metamodell der RESTful Services* zeigt Abbildung 5.6.

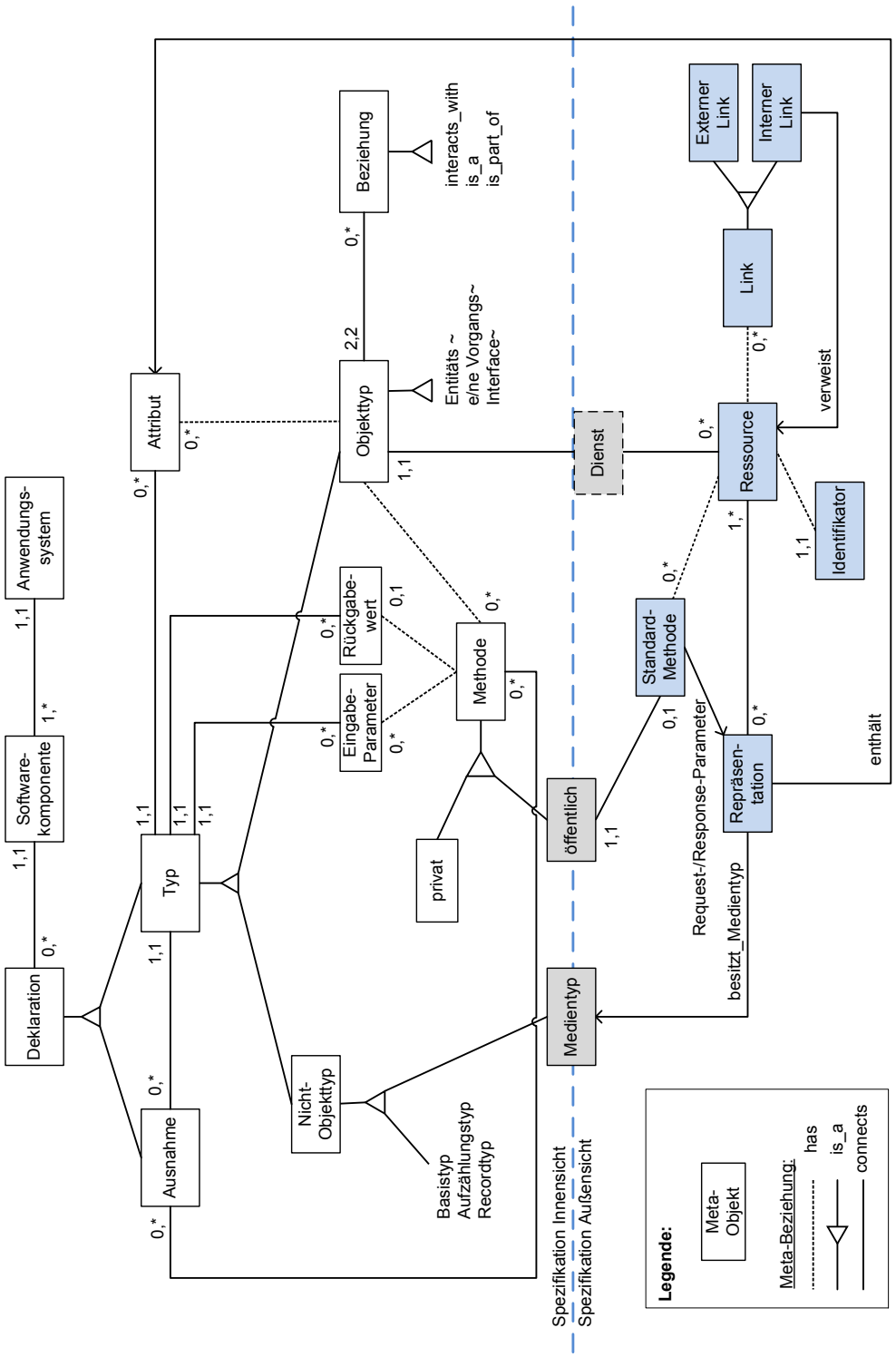


Abbildung 5.6: Metamodell der RESTful Services
(Erweiterung des Metamodells von [Mali97, S. 74], eigene Darstellung)

Das Metamodell zur softwaretechnischen Spezifikation objektorientierter Anwendungssysteme nach MALISCHEWSKI [Mali97, S. 74] bildet die Grundlage des Metamodells der RESTful Services. In seiner Arbeit erweitert der Autor das SOM-Metamodell der fachlichen AwS-Spezifikation um Konzepte zur Typisierung und Fehlerbehandlung sowie zusätzliche softwaretechnische Komponenten für den Aufbau einer Softwarearchitektur. Dieses SOM-Metamodell der Softwareebene wird in der vorliegenden Arbeit um eine servicebezogene Spezialisierung der

Objekttypen (EOT, eVOT und neVOT), das Medientyp-Metaobjekt, die Metaobjekte der Außensicht sowie die Beziehungen zwischen den Bausteinen der Innen- und Außensicht erweitert.

Aus der *Innensicht* (weiße Füllfarbe) bildet eine Menge von *Objekttypen*, die entweder EOT, eVOT, neVOT oder IOT sind, den Kern des AwS (Abschnitt 5.3.2). Die Objekttypen spezifizieren Attribute und Methoden, und stehen mit anderen Objekttypen in *Beziehung*. *Methoden* besitzen ein bis beliebig viele *Eingabeparameter* und optional einen *Rückgabewert*. Einer Methode sind zudem beliebig viele *Ausnahmen* zuordenbar, die bei der fehlerhaften Ausführung der Methode ausgelöst werden können. *Nicht-Objekttypen* dienen der Typisierung von Attributen, Eingabeparametern, Rückgabewerten und Ausnahmen. Des Weiteren besitzen Nicht-Objekttypen keine Objektidentität und erlauben die Typisierung von Elementen nach den zugrundeliegenden Konzepten der eingesetzten Programmiersprache. Ausnahmen und Typen sind *Deklarationen*, die in einer *Softwarekomponente* des *Anwendungssystems* gekapselt werden (in Anlehnung an [Mali97, S. 73 ff.]).

Die Funktionalität der RESTful SOA wird aus der *Außensicht* als Menge von *Ressourcen* mit einheitlicher Schnittstelle beschrieben (hellblaue Füllfarbe). Entsprechend der vorgeschlagenen Klassifizierung von Services erfolgt eine Spezialisierung von Ressourcen als *Entitäts-* oder (*e*/*ne*) *Vorgangsressourcen* (Abschnitt 3.3.3.1). Jede Ressource besitzt genau einen *Identifikator* (URI) und ist über ihre *öffentlichen Standardmethoden*[29] zugreifbar, die aus der Innensicht öffentliche Methoden von Objekttypen darstellen. Im Rahmen der Interaktion (Request-/Response-Nachrichten) mit Ressourcen werden Zustandsbeschreibungen in Form von *Repräsentationen* ausgetauscht. Diese beziehen sich auf Attribute korrespondierender Objekttypen. Zur Typisierung der Informationen von Repräsentationen werden *Medientypen* verwendet, welche damit Content Negotiation ermöglichen. Innerhalb der RESTful SOA spezifizieren *interne Links* die Beziehungen zwischen Ressourcen. *Externe Links* verweisen auf Komponenten der Umwelt. Interne und externe Verlinkungen spannen die Hypermedia des verteilten Systems auf (vgl. RESTful Services in Abschnitt 3.3.2).

Den *Übergang zwischen Innen- und Außensicht* (graue Füllfarbe) der RESTful SOA markieren der Medientyp, die öffentlichen Methoden von Objekttypen sowie das Dienst-Konzept. Die einheitliche Ressourcen-Schnittstelle spezifiziert zum einen die URI (Identifikator) für den Zugriff auf die definierten HTTP-Operationen (Standard-Methoden) und definiert zum anderen, welche Medientypen einer Repräsentation dabei konsumiert oder produziert werden.

5.3.4 Metamodell des Ressourcenschemas

Das Metamodell des Ressourcenschemas dient der detaillierten Beschreibung der Nutzerschnittstelle des AwS. Das Metamodell wird hierbei als Projektion auf das integrierte Metamodell der Ebene REST-Architekturmodell verstanden. Das Ziel des zu konzipierenden Metamodells ist die Definition der Modellbausteine, die zur Spezifikation der einheitlichen Ressourcenschnittstelle erforderlich sind. Die Kernbausteine des Ressourcenschemas wurden im Meta-

[29] Das Standardprotokoll zur Realisierung der Kommunikation zwischen Komponenten einer RESTful SOA ist HTTP (Abschnitte 3.3.2 und 3.4.2). Deshalb werden im weiteren Verlauf der Arbeit die Begriffe *HTTP-Methode* und *öffentliche Standardmethode* gleichbedeutend verwendet.

modell der RESTful Services bereits vorgestellt. Das *Metamodell des Ressourcenschemas* zeigt Abbildung 5.7.

Das Ressourcenschema spezifiziert die einheitliche Schnittstelle jeder Ressource mit den Eigenschaften *Identifikator* (URI), *Standard-Methoden* (HTTP) sowie den ausgetauschten *Repräsentationen*. Der Aufruf einer HTTP-Methode erfolgt durch den Empfang einer *Request*-Nachricht (Aufruf), die in ihrem Nachrichtenkörper eine Repräsentation enthalten kann und von der Methode konsumiert wird. Als Antwort auf einen Aufruf sendet die Methode eine *Response*-Nachricht, die optional eine Repräsentation enthält, um diese an den Client zu übermitteln. Zur Spezifikation von Verknüpfungen auf interne oder externe Komponenten definieren Ressourcen *Links*, die der Umwelt als Bestandteil der Repräsentation bereitgestellt werden können. Die Beziehung zwischen Ressource und ihren Attributen sowie der Repräsentation eines Dienstes sind über die zugehörigen Objekttypen bzw. Ressourcen transitiv beschreibbar (Abschnitt 3.3.2.1).

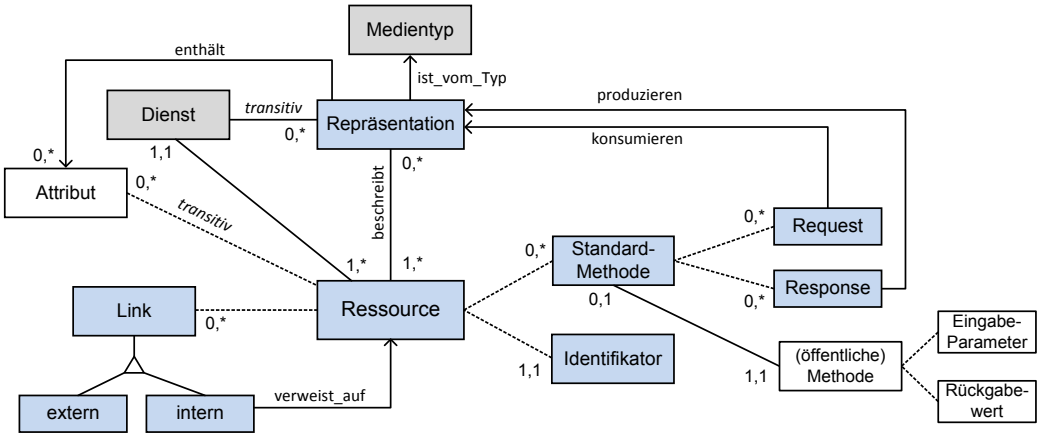


Abbildung 5.7: Metamodell des Ressourcenschemas

Zur Darstellung von Ressourcenschemata wird in dieser Arbeit eine tabellarische Form gewählt, da Schnittstellenbeschreibungen meist als Auflistung einer Vielzahl von Ressourcen und ihrer Eigenschaften genutzt werden. Die Tabellenstruktur bietet zudem eine gute Übersichtlichkeit und lässt sich aufgrund ihrer Gliederung einfach in andere menschen- oder maschinenlesbare Dokumentationsformate überführen (z. B. die Web Application Description Language (WADL) [Hadl09]).

In der REST-Literatur existiert bereits eine Reihe von Vorschlägen zur Beschreibung der Schnittstelle von REST-Anwendungen (z. B. [TaVa13], [Schr11], [VaPa09]). Häufig steht dabei die formale Spezifikation der Schnittstelle im Fokus. Die Konzeption in diesen Ansätzen zielt somit nicht auf eine integrierte Betrachtung der Außen- und Innensicht von RESTful SOA sowie die Einbeziehung von höheren Abstraktionsebenen in einem einheitlichen Begriffssystem ab. Die Untersuchung und Diskussion verwandter Modellierungsansätze erfolgt in Abschnitt 8.1.2.

5.3.5 Metamodelle der Daten- und Dokumentenschemata

Die Nutzung der durch die RESTful SOA bereitgestellten Geschäftsfunktionalität basiert auf der Verwaltung *transienter* und *persistenter Ressourcenzustände*. Eine wichtige Anforderung an

den Austausch von Zustandsinformationen ist die REST-Bedingung der *Zustandslosigkeit* (Abschnitt 3.3.2.3).

Die Wahl der Metamodelle zur Spezifikation der Datenhaltungs- und Dokumentenschemata auf der Ebene der Softwarearchitektur ist grundlegend vom gewählten Datenbankverwaltungssystem bzw. den verwendeten Dokumentformaten auf der Implementierungsebene abhängig. Im Gegensatz zur Spezifikation der RESTful Services erfordert die Konzeption der Modellierungsansätze für die Daten- und Dokumentenschemata stets eine Berücksichtigung der auf der plattformspezifischen Ebene unterstützten Daten- bzw. Dokumentmodelle.

Damit die Entwicklungsschritte zur Modellierung von Daten- und Dokumentenschemata konkret gestaltet und die Anwendbarkeit der SOM-R-Methodik im Rahmen der Fallstudie demonstriert werden kann, wird folgende Annahme getroffen: Die Datenhaltung des AwS *RESTful SOA* wird durch ein relationales Datenbanksystem (DBS) und der Dokumentenaustausch auf Basis von JSON realisiert. Die Beschreibung von persistenten und transienten Daten der RESTful SOA und die Verknüpfung der hierbei verwendeten Konzepte mit dem Begriffssystem der REST-Ebene wird somit am Beispiel zweier konkreter Modellierungsansätze demonstriert.

MODELLIERUNG DER DATENSCHEMATA

Datenbankschemata werden auf konzeptueller Ebene in Form von Datenschemata spezifiziert, deren verfügbare Bausteine und Beziehungen in einem *Datenmodell* definiert sind [FeSi13, S. 150]. Ein etablierter Standard zur Realisierung der Datenverwaltungsebene von AwS ist das *relationale Datenbankmodell* bzw. *Relationenmodell* [Codd70]. Dieses wird auf konzeptueller Ebene meist durch Datenmodellierungsansätze wie das Entity-Relationship-Modell (ERM) [Chen76] oder das Strukturierte Entity-Relationship-Modell (SERM) ([Sinz87], [Sinz88b]) beschrieben. Auf der Ebene der Implementierung unterstützt eine Vielzahl an Datenbankverwaltungssystemen (DBVS) die relationale Datenhaltung (z. B. [Gabr13], [KeEi13], [Voss08]).

Alternativ zum Relationenmodell bieten sich für die Entwicklung der Datenhaltungsebene der RESTful SOA auch objektorientierte, objektrelationale oder, die in jüngster Zeit diskutierten dokumentenorientierten Datenbankmodelle an (z. B. [KeEi13], [Edli11], [Voss08]). In aktuellen Arbeiten zur Datenhaltung in verteilten AwS wird zudem eine Reihe von Technologien unter der Bezeichnung *NoSQL*[30] zusammengefasst. NoSQL-Datenbanksysteme kamen im Zuge der Entwicklung von Web-Anwendungen und ihren Anforderungen, insbesondere nach hoher Skalierbarkeit, auf ([LaCr13, S. 97 ff.], [FeSi13, S. 414 f.], [Edli11, S. 1 ff.]).

Das OS-ED bildet den Ausgangspunkt für die Ableitung des Datenschemas in der SOM-R-Methodik. Die Integration des Datenmodells in die weiteren Metamodelle des REST-Architekturmodells findet über die Beziehung zwischen den Bausteinen EOT und DOT statt. EOTs greifen auf DOTs zu und definieren dadurch die Speicherung persistenter Daten von Diensten (Abbildung 5.5). Aufgrund seiner objektorientierten Form ist das OS-ED grundsätzlich unabhängig vom eingesetzten Datenmodell und der Spezifikation der Datenhaltungsebene. Die

[30] Eine Übersicht aktuell verfügbarer NoSQL-Datenbanksysteme findet sich unter http://nosql-database.org/ (Abruf am 24. Februar 2015).

Überführung der EOTs und ihrer Beziehungen ist sowohl in die Datenobjekttypen (und Beziehungen) von relationalen, objektorientierten, als auch objektrelationalen Datenschemata möglich. Das OS-ED kann auch als Quelle für die Ableitung der Modelle von NoSQL-Datenbanksystemen dienen (z. B. [WoBe13, S. 6 ff.]).

Als relationales Datenmodell wird das *Strukturierte Entity-Relationship-Modell* (SERM) [FeSi13, S. 158 ff.] im weiteren Verlauf der Arbeit zur Gestaltung der Datensicht eingesetzt. Das SERM-Metamodell wird in Anhang B.1 vorgestellt.

Modellierung der Dokumentstruktur

Dokumenttypen definieren die Struktur von Zustandsbeschreibungen, die Ressourcen innerhalb der RESTful SOA produzieren oder konsumieren. Im Umfeld von REST-Anwendungen haben sich Standards auf Basis von JSON und XML als verbreitete Repräsentationsformate etabliert ([Tilk11, S. 204 f.], [LaGü12, S. 26], [MPD10, S. 112 f.]).

Dokumenttypen beschreiben Daten in strukturierter Form als Menge von *Attributen* und ihren zugehörigen *Datentypen* [WoBe13, S. 7]. Je nach Art des verwendeten Repräsentationsformats handelt es sich dabei um einfache oder komplexe Datentypen (Abschnitt 3.4.3, JSON). Die Definition von Dokumenttypen basiert auf der Syntax des gewählten Dokumentformats und erfolgt in der Arbeit aus Gründen der einfachen Darstellung in textueller Form. Zudem bietet sich die Nutzung von Strukturierungskonzepten des hierfür eingesetzten Spezifikationsstandards an.

Wie bereits in Abschnitt 3.4.3 erläutert, wird im weiteren Verlauf der Arbeit JSON als Standardformat für Dokumente verwendet. Ein JSON-Metamodell wurde bereits eingeführt (Abbildung 3.9).

5.3.6 Erweitertes Metamodell der BPMN

Die Realisierung der RESTful Vorgangsservices wird aus der Innensicht durch das Lösungsverfahren der (elementaren/nicht-elementaren) VOTs als gesteuerten Ablauf von Aktionen beschrieben (Abschnitt 3.3.3.1, [KrSi11, S. 295 f.]). Für die Spezifikation des VOT-Lösungsverfahrens wird nachfolgend eine Notationssprache vorgestellt, welche die Komposition von RESTful Services in Form eines Workflows[31] modelliert (z. B. [FrRü12, S. 187 ff.], [OMG11a], [OASI07]).

Als Workflowsprache wird in der vorliegenden Arbeit die *Business Process Model and Notation* [OMG11a], kurz BPMN, verwendet. Die BPMN wurde im Jahre 2011 als offizieller OMG-Standard in der Version 2.0 verabschiedet [OMG11a, S. 1] und verfolgt eine Reihe von Zielen, die für den Einsatz in der SOM-R-Methodik sprechen:

- *Leicht verständliche Sprache zur grafischen Modellierung von Prozessen*. Die BPMN 2.0 soll von Fachanwendern und Systementwicklern auf unterschiedlichen Abstraktionsebenen gleichermaßen einsetzbar sein. Hierfür werden eine Notation und ein Meta-

[31] Als *Workflow* wird allgemein die Durchführung eines Vorgangs bezeichnet [FeSi13, S. 103 f.].

modell bereitgestellt, die in das Begriffssystem der SOM-Methodik übernommen werden können (z. B. [PüSi10], [FeSi13, S. 190 ff.]).

- *Modellierungsstandard zur Visualisierung unterschiedlicher Workflowsprachen*. Die BPMN 2.0 definiert zur Erreichung des ersten Ziels ein XML-Serialisierungsformat, auf dessen Basis eine Überführung zwischen BPMN-Schemata und alternativen XML-basierten Workflowsprachen spezifiziert werden kann ([OMG11a], [FeSi13, S. 190]). Ein Beispiel hierfür ist die Transformation von BPMN in WS-BPEL[32] [OMG11a, S. 445 ff.].
- *Automatisierte Ausführung von Prozessmodellen*. Die BPMN 2.0 ermöglicht die Detaillierung bzw. Anreicherung ihrer Schemata mit technischen Attributen (z. B. [FrRü12, S. 156 ff.]). Auf Basis BPMN 2.0-konformer Prozess-Engines[33] findet in der Praxis vermehrt die automatisierte Ausführung von BPMN-Schemata statt. Durch diesen Ansatz ist es möglich, die semantische Lücke zwischen Ablaufspezifikationen auf fachlicher und technischer AT-Ebene, zumindest teilweise, zu überbrücken.

Im REST-Architekturmodell spezifiziert das Lösungsverfahren eines VOT die Koordination einer Menge von Entitäts- und/oder Vorgangsdiensten bzw. deren Ressourcen bei der Durchführung von betrieblichen Aufgaben. Zur grafischen Darstellung der einheitlichen REST-Schnittstelle wird deshalb die Erweiterung des BPMN-Metamodells, um die Konzepte *REST-Aktivität*, *REST-Nachrichtenfluss* sowie *REST-Schnittstelle* vorgenommen. Das erweiterte BPMN-Metamodell visualisiert Abbildung 5.8.

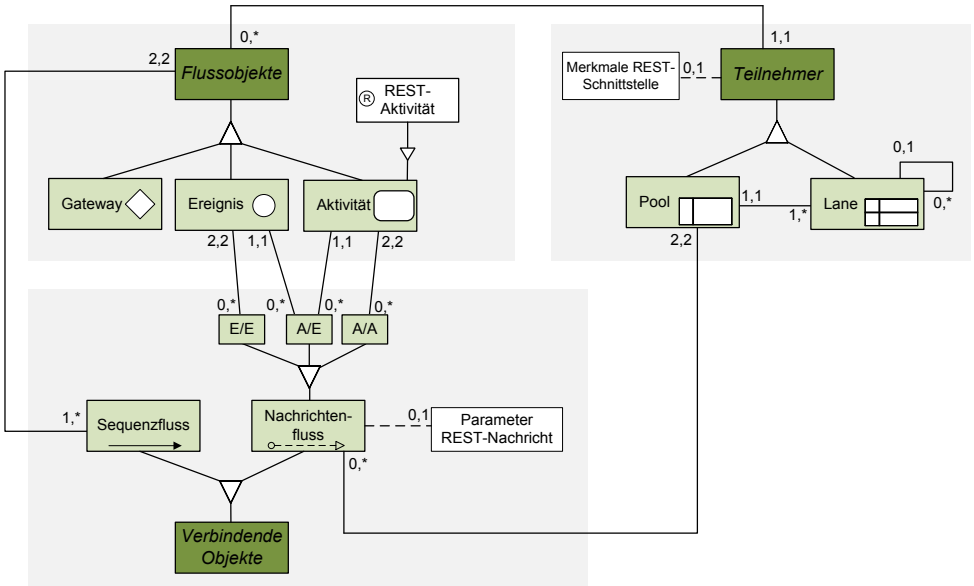


Abbildung 5.8: Metamodell der BPMN (in Anlehnung an [FeSi13, S. 192], eigene Erweiterung)

Die neuen REST-spezifischen Elemente (weiße Füllfarbe) werden als Spezialisierungen oder Eigenschaften von Metaobjekten des systemorientierten Metamodells nach FERSTL und SINZ

[32] *WS-BPEL* (*Web Services-Business Process Execution Language*) [OASI07] ist eine Sprache zur Orchestrierung von Web-Services [FeSi13, S. 441 f.]. Ein (WS-)BPEL-Prozess wird wiederum selbst als Web-Service in einer BPEL-Engine ausgeführt.

[33] Beispiele für aktuelle Open-Source-Entwicklungen von BPMN-Engines sind *Activiti* (activiti.org), *Camunda* (camunda.org) oder *jBPM* (jbpm.jboss.org) (Abruf am 25. März 2015).

eingeführt und erweitern dieses punktuell [FeSi13, S. 192]. Hierdurch wird eine konsistente Abstimmung der RESTful BPMN auf das Begriffssystem der SOM-R-Methodik gefördert.

Ein BPMN-Diagramm besteht aus Elementen der Kategorien *Flussobjekte* (Flow Objects), *verbindende Objekte* (Connecting Objects) und *Teilnehmern* (Swim Lanes). Bestandteile der Kategorie Flussobjekte sind *Aktivitäten* (Activity), *Ereignisse* (Event) und *Gateways*. Verbindende Objekte verknüpfen Flussobjekte durch Sequenzflüsse und Nachrichtenflüsse sowie Assoziationen (nicht dargestellt). Teilnehmer innerhalb von Prozessmodellen werden als *Swim Lanes* in Form von *Pools* und *Lanes* repräsentiert[34] ([FrRü12, S. 21], [FeSi13, S. 192 f.]).

Flussobjekte sind die Hauptbestandteile zur Spezifikation des Verhaltens im Prozessmodell (Ausführungen nach [FrRü12, S. 19 ff.], [FeSi13, S. 190 ff.]):

- *Aktivitäten* beschreiben jeweils eine zu verrichtende Tätigkeit oder einen Verarbeitungsschritt innerhalb des Prozesses und stellen das Kernkonzept der BPMN dar.
  RESTful SOA: Eine Aktivität wird als REST-Aktivität markiert, wenn sie den Empfang oder das Senden einer HTTP-Nachricht (REST-Nachrichtenfluss) durchführt und demnach einen öffentlichen Operator auslöst.
- *Gateways* spezifizieren den Prozessablauf im Form exklusiver (Durchführung genau eines Pfades), inklusiver (mindestens ein Pfad) oder paralleler (alle Pfade) Verzweigungen sowie entsprechender Synchronisationsarten bei der Zusammenführung von Prozesspfaden.
- *Ereignisse* definieren Zustände, die den Prozessablauf beeinflussen. Es wird grundsätzlich zwischen eintretenden („catching events“) und ausgelösten Ereignissen („throwing events“) unterschieden. Zudem lässt sich ein Ereignis in Startereignis, Zwischenereignis und Endereignis unterscheiden.

Teilnehmer sind logische Einheiten, die den Prozessfluss hierarchisch steuern und kontrollieren (Ausführungen nach [FrRü12, S. 22 und S. 44 ff.], [OMG11a, S. 112 ff.]):

- *Pools* beschreiben genau einen Teilnehmer (Participant) im Prozess.
  Im Kontext von *RESTful SOA* handelt es sich um *interne* oder *externe Komponenten*, wie Dienste bzw. Clients. Teilnehmer verschiedener Prozesse interagieren über den Austausch von Nachrichten miteinander.
- Innerhalb eines Pools können die Zuständigkeiten für einen Teil des Prozessmodells mit Hilfe von *Lanes* zugeordnet werden, die häufig personelle oder maschinelle Aufgabenträger für die Durchführung bestimmter Aktivitäten beschreiben.

Teilnehmer repräsentieren im REST-Architekturmodell *Vorgangsdienste*, denen die Merkmale der einheitlichen REST-Schnittstelle optional als Eigenschaft zugeordnet werden können. *REST-Eigenschaften* sind z. B. URI, veröffentlichte Methoden oder akzeptierte Medientypen. Der Empfang oder das Senden von HTTP-Nachrichten durch Methoden realisiert eine REST-Aktivität.

[34] Zur Vermeidung einer Vermischung deutscher und englischer Bezeichnungen werden im Folgenden die deutschen Begriffe für BPMN-Elemente verwendet (Übersetzung nach [FrRü12, S. 21-25]). Ausgenommen hiervon sind die Begriffe *Gateway*, *Pool* und *Lane*, da die Semantik dieser Elemente durch die Originalbezeichnung treffender transportiert wird.

Verbindende Objekte beschreiben Beziehungen zwischen Flussobjekten innerhalb eines oder zwischen verschiedenen Teilnehmern [FrRü12, S. 27]:

- *Sequenzflüsse* verbinden Flussobjekte innerhalb des Pools eines Prozessmodells und bringen diese in eine zeitlich-logische Reihenfolge [OMG11a, S. 97 f.].
- *Nachrichtenflüsse* verknüpfen unterschiedliche Pools miteinander. Dabei beschreiben sie den Nachrichtenaustausch entweder zwischen Flussobjekten innerhalb einzelner Pools und/oder starten bzw. enden an der Poolgrenze [OMG11a, S. 120].
  RESTful SOA: Der Nachrichtenaustausch zwischen (internen) Diensten und externen Nutzern der RESTful SOA soll durch das Verknüpfen der Poolgrenze des externen Teilnehmers mit dem Flussobjekt des internen Teilnehmers modelliert werden.
  Nachrichtenflüssen, die mit der Übermittlung einer HTTP-Anfrage oder -Antwort betraut sind, kann die Spezifikation der ausgetauschten *REST-Nachricht* als Eigenschaft zugeordnet werden (Abschnitt 3.4.2).

Als weitere Basiskategorien definiert der BPMN-Standard *Artefakte* (Artifacts) und *Daten* (Data). Daten werden durch Datenobjekte, Dateninputs, Datenoutputs und Datenspeicher repräsentiert. Artefakte sind (textuelle) Annotationen oder Gruppierungen [OMG11a, S. 27 f.]. Artefakte und Daten werden im eingeführten BPMN-Metamodell nicht explizit dargestellt, da sie für die Modellierung von Lösungsverfahren im weiteren Verlauf der Arbeit nicht eingesetzt werden. Es ist jedoch möglich, die Zuordnung zwischen REST-Eigenschaften und -Elementen als textuelle Annotation über eine Assoziations-Beziehung darzustellen.

Die Erweiterung des BPMN-Metamodells dient in dieser Arbeit der Spezifikation der Kompositionslogik von RESTful Services. Die Anpassung der BPM-Notation wird dazu möglichst minimalistisch vorgenommen, sodass ein REST-BPMN-Schema keine neuen Kernelemente aufweist. Ziel dieser Vorgehensweise ist, dass die in der SOM-R-Methodik spezifizierten BPMN-Schemata weiterhin konform zur Standardnotation sind und so mit gängigen BPMN-Engines ausführbar bleiben. In der aktuellen Literatur existieren bereits Vorschläge zur RESTful Erweiterung von Workflowsprachen (z. B. [Over08], [CDK+07], [Paut11]). Jedoch beschränken sich die Ansätze auf die Einführung von REST-Notationselementen. Die explizite Definition eines Metamodells erfolgt dagegen nicht und erschwert damit eine Integration in das SOM-R-Architekturmodell. Eine genauere Betrachtung von verwandten Arbeiten aus dem Bereich der RESTful Modellierung mit BPMN erfolgt in Abschnitt 8.1.3.

5.4 Konzeption des SOM-R-Vorgehensmodells

Die Überführung des SOM-Geschäftsprozessmodells in die Implementierung der RESTful SOA erfolgt in *sechs Entwicklungsschritten*, die den Abstraktionsgrad des Modellsystems bis zur Zielebene des Implementierungsmodells sukzessive reduzieren (Abbildung 5.1 und Abbildung 5.3). Das Vorgehen zur modellbasierten Spezifikation von RESTful SOA wird nachfolgend zunächst in Bezug auf die Aufgabe der Systementwicklung in einem phasenorientierten Vorgehensmodell beschrieben. Anschließend wird der Ablauf zur Spezifikation der Bestandteile des SOM-R-Architekturmodells genauer erläutert.

PHASENORIENTIERTES VORGEHEN DER MODELLBASIERTEN SERVICE-SPEZIFIKATION

Die phasenorientierte Durchführung der Entwicklungsaufgabe sieht die schrittweise Überbrückung der semantischen Lücke zwischen den vier Beschreibungsebenen der Systementwicklung vor [FeSi13, S. 499 f.]. Als Hauptphasen werden *Anforderungsanalyse und -definition* (A), *Softwaredesign* (D) und *Realisierung* (R) durchlaufen, die das Lösungsverfahren der Systementwicklungsaufgabe in Form von Aktivitäten spezifizieren. Das Ergebnis jeder Aktivität bildet den Input der nachfolgenden Aktivität (Abschnitte 2.2.2 und 2.2.3). Die Beziehungen zwischen Ebenen und den Ablauf der Aktivitäten veranschaulicht Abbildung 5.9.

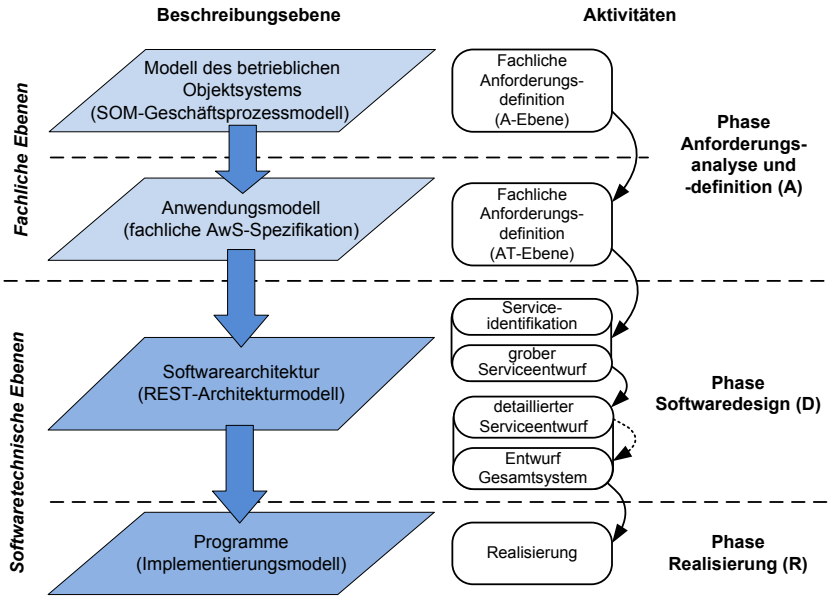


Abbildung 5.9: Phasenorientiertes Vorgehen und Kernaktivitäten der Serviceentwicklung (in Anlehnung an [FeSi13, S. 499], eigene Anpassungen)

Die Phase *Anforderungsanalyse und -definition* umfasst zwei Teilaktivitäten, welche die Beschreibung der *fachlichen Anforderungen* an das AwS auf Aufgaben- und Aufgabenträgerebene (A- bzw. AT-Ebene) beinhalten. Die Durchführung beider Aktivitäten entspricht demzufolge der Gestaltung der Modellebenen von SOM-Geschäftsprozessmodell und fachlicher AwS-Spezifikation sowie der Modellebenen-Beziehung zwischen diesen [FeSi13, S. 499].

Das Anwendungsmodell ist Ausgangspunkt für die (initiale) Identifikation und anschließende Spezifikation von Diensten der RESTful SOA. Das Vorgehen zur Durchführung der Phase *Softwaredesign* besteht aus insgesamt vier Aktivitäten. Die initiale *Service-Identifikation* kennzeichnet den Übergang auf die softwaretechnische Ebene. Sie ist Voraussetzung für den *groben Serviceentwurf* und wird aufgrund der gegenseitigen Abhängigkeiten gemeinsam mit diesem durchgeführt. Der grobe Serviceentwurf ist wiederum Input für die schrittweise Überarbeitung und Detaillierung der Service-Spezifikation im *detaillierten Serviceentwurf*. Das Ergebnis der Phase Softwaredesign bildet der *Entwurf des Gesamtsystems*. Die Aktivitäten der D-Phase orientieren sich an Vorschlägen für SOA-Entwicklungsvorgehen in der Literatur (z. B. [Josu08, S. 169 ff.], [Müll07, S. 145 ff.]).

Die detaillierte Spezifikation des Gesamtsystems geht in die Ableitung des plattformspezifischen Implementierungsmodells ein, das die *Realisierung* der RESTful SOA beschreibt.

SOM-R-VORGEHENSMODELL

Das phasenorientierte Vorgehen leitet die systematische Durchführung der Systementwicklungsaufgabe top-down (von oben nach unten) entlang der vier Modellebenen des SOM-R-Architekturmodells als Folge von Schema-Transformationen (T1-T3) und Schema-Konsolidierungen (K1-K3) an (Abbildung 5.1). Das grundlegende Vorgehen zur Erstellung dieser Schemata beschreibt das SOM-R-Vorgehensmodell, welches in Abbildung 5.10 dargestellt ist. Der Ablauf der Teilschritte von Transformationen und Konsolidierungen des Entwicklungsvorgehens wird in Abschnitt 5.5 detailliert vorgestellt und erläutert.

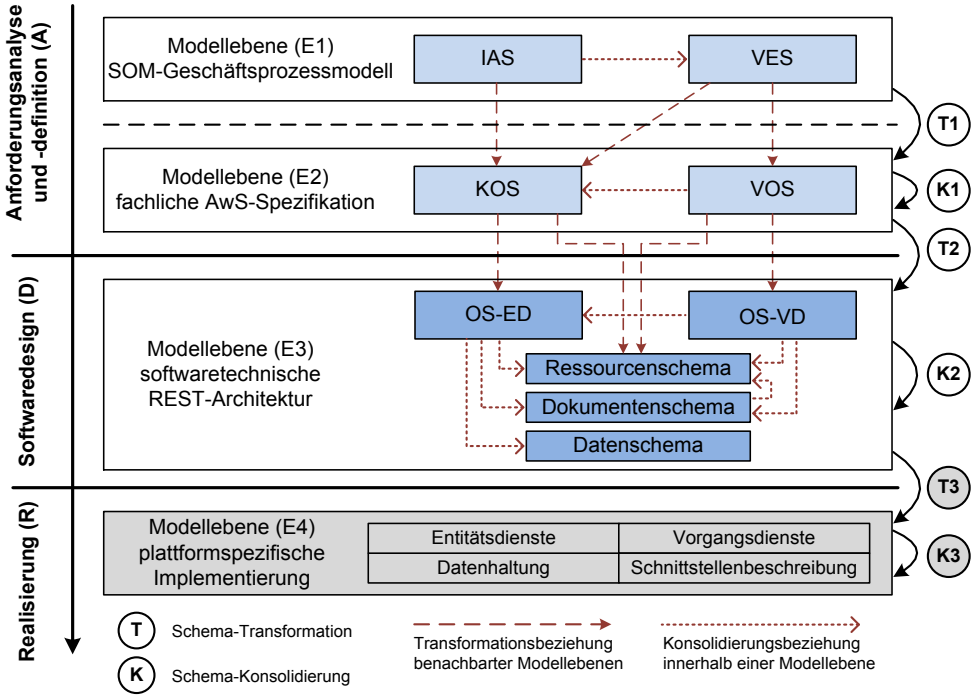


Abbildung 5.10: Vorgehensmodell der SOM-R-Entwicklungsmethodik

Die Modellebenen *Geschäftsprozessmodell* und *fachliche AwS-Spezifikation* werden entsprechend dem SOM-Vorgehensmodell entwickelt [FeSi13, S. 197 ff.]. Ausgehend vom IAS der Struktursicht wird das korrespondierende VES in der Verhaltenssicht modelliert. IAS und VES bilden den Input für die erste Schema-Transformation (T1) zur initialen Ableitung der fachlichen Spezifikation der Aufgabenträgerebene.

Das IAS sowie die ablaufspezifischen Informationen des VES dienen als Ausgangspunkt für die Ableitung des initialen KOS. Das VES wird direkt in das initiale VOS transformiert. Die Überarbeitung und Anreicherung der initialen Schemata[35] KOS und VOS erfolgt in der ersten Schema-Konsolidierung (K1). Den Objekttypen des VOS wird hierbei ein Ausschnitt des überarbeiteten (konsolidierten) KOS zugeordnet und die Interface-Objekttypen zur Umwelt werden aufgedeckt. Dieses Vorgehen korrespondiert mit der Durchführung der Entwicklungsaktivitäten *Definition der fachlichen Anforderungen auf der A-Ebene* bzw. *AT-Ebene*.

[35] Die Unterscheidung zwischen initialen und konsolidierten Schemata der Modellebenen E2 und E3 wird in Abbildung 5.10 aus dem Grund einer übersichtlicheren Darstellung nicht visualisiert.

Die überarbeiteten Schemata KOS und VOS sind Input für die zweite Schema-Transformation (T2), die den Übergang auf die softwaretechnische Ebene markiert. Ergebnis von T2 sind die initialen Schemata *OS-ED* (Objektschema der Entitätsdienste), *OS-VD* (Objektschema der Vorgangsdienste) sowie das initiale *Ressourcenschema*. Die Objekttypen von KOS und VOS der fachlichen AwS-Spezifikation bilden die Quelle für eine Identifikation und Ableitung von Entitäts- bzw. Vorgangsdiensten und deren Ressourcen. Die Schnittstellenbeschreibung wird im Ressourcenschema mit den initial abgeleiteten Ressourcen erfasst. Die Ableitung der Modellebene korrespondiert mit den Aktivitäten *Service-Identifikation* und *grober Service-Entwurf*. Das initiale OS-ED und initiale OS-VD werden in der zweiten Schema-Konsolidierung (K2) aus Außen- und Innensicht strukturell überarbeitet und um softwaretechnische Merkmale angereichert. Auf dieser Basis erfolgen die Definition der Datenhaltungsstruktur und der Struktur transienter Dokumente sowie die finale Spezifikation der Benutzerschnittstelle und Dienstbeschreibung im Ressourcenschema. Die Durchführung der Aktivitäten *detaillierter Service-Entwurf* und *Entwurf des Gesamtsystems* erfolgt somit in K2.

Die überarbeiteten Schemata des OS-ED, OS-VD sowie Daten- und Dokumentstruktur und Ressourcenschema gehen als Input in die dritte Schema-Transformation (T3) ein. Das Ziel dieses letzten Transformationsschrittes ist eine modellgetriebene und damit automatisierte Ableitung der Implementierung der RESTful SOA (Implementierungsmodell) aus dem softwaretechnischen REST-Architekturmodell. Im (optionalen) Entwicklungsschritt der dritten Schema-Konsolidierung (K3) kann das Implementierungsmodell bei Bedarf um fehlende Implementierungslogik und technische Anforderungen der Basismaschine angereichert werden, die im Zuge der dritten Transformation nicht spezifizierbar waren. Die Schritte T3 und K3 korrespondieren mit der Durchführung der Aktivität *Realisierung*. Eine Ausgestaltung von T3 und K3 erfordert die Festlegung konkreter Technologien und Basismaschinen auf der Implementierungsebene, weshalb im Vorgehensmodell der Methodik von einer detaillierten Darstellung abstrahiert wird (graue Füllfarbe).

ANMERKUNGEN ZUM ENTWICKLUNGSVORGEHEN

Das Vorgehensmodell der Entwicklungsmethodik sieht die Spezifikation der RESTful SOA grundsätzlich entlang der Abstraktionsebenen des Architekturmodells von „oben nach unten“ vor (top-down)[36]. Mit der Wahl des *Top-Down-Ansatzes* wird angestrebt, dass eine Begründung der RESTful Services stets auf Basis eines (modellierten) fachlichen Bedarfs erfolgt. Ziel ist zudem, eine enge und konsistente Abstimmung der abgeleiteten Implementierung mit den fachlichen Anforderungen des SOM-Geschäftsprozessmodells über die Beschreibungsebenen des Systems hinweg zu unterstützen.

Der Top-Down-Charakter der modellbasierten Systementwicklung wird in einem Projekt jedoch nicht immer vollständig umsetzbar sein. Beispielsweise können aufgrund der nicht-

[36] Ein Top-Down-Vorgehen wird von ERL als vorteilhaft zur konsequenten Umsetzung des SOA-Konzepts angesehen [Erl08b, S. 518 f.]. Im Kontext der Bestimmung von Diensten auf Basis von Altsystemen ist dagegen der Einsatz von Bottom-Up-Ansätzen von Bedeutung (z. B. [Mahl07, S. 169 ff.]). In der Literatur wird jedoch zudem häufig die Verwendung eines kombinierten Vorgehens zur Entwicklung von SOA empfohlen (z. B. [Josu08, S. 106], [Erl08b, S. 518 f.], [BlSc06, S. 173-175]).

automatisierbaren Entwurfsschritte der Konsolidierung die Ergebnisse der aktuell ausgeführten Aktivität die wiederholte Durchführung einer früheren Aktivität notwendig machen (iterativ). Ein *iteratives Vorgehen* ist damit zur Abstimmung von z. B. Realisierung und REST-Architekturmodell verwendbar. Die Erkenntnisse über die implementierte Architektur des AwS könnten zudem in einem Bottom-Up-Schritt als Input einer wiederholten Überarbeitung der detaillierten Dienstspezifikation genutzt werden.

Im Vorgehen zur Entwicklung der erweiterten SOM-Architektur stellt das Top-Down-Vorgehen den idealtypischen Fall dar, von dem in praxisbezogenen Entwicklungsprojekten bei Bedarf auch abgewichen werden kann (Abschnitt 4.1.1). Die finale Abstimmung des Gesamt-Modellsystems erfolgt jedoch immer von oben nach unten [FeSi13, S. 199]. Die Entwicklungsschritte der Modellbildung werden im nachfolgenden Abschnitt erarbeitet.

5.5 Konzeption der Entwicklungsschritte für die Modellbildung

Die Konzeption des Lösungsverfahrens zur Modellbildung auf den einzelnen Ebenen des SOM-R-Architekturmodells ist Gegenstand dieses Abschnitts. Das Vorgehen der Modellbildung entspricht dabei einer schrittweisen Durchführung der Systementwicklungsaufgabe und wurde im SOM-R-Vorgehensmodell bereits grob eingeführt (Abbildung 5.10).

Eine konkrete Ausgestaltung der Entwicklungsschritte setzt die Bestimmung der zu automatisierenden Teile der Systementwicklungsaufgabe voraus. Hier wird zwischen zwei Typen von Entwicklungsschritten unterschieden (Abschnitte 5.1.1 und 5.4):

- Die Entwicklungsschritte der *Transformation* sind *vollständig automatisiert* durchführbar und kapseln alle konzeptionell begründbaren Abhängigkeitsbeziehungen zwischen Modellelementen. Die Überführung von Metaobjekten benachbarter Modellebenen wird auf Basis metamodellbasierter Schema-Transformationen spezifiziert.
- Die Entwicklungsschritte der *Konsolidierung* kapseln alle *nicht-automatisierten* Entwurfsentscheidungen zur Gestaltung einer Modellebene und der Spezifikation ihrer Elemente. Das Vorgehen zur Modellbildung sieht die methodische Anleitung der Entwicklung durch die Definition von Überarbeitungsschritten sowie die Bereitstellung weiterer Hilfsinformationen vor.

Die Modellbildung wird grundsätzlich unabhängig von konkreten Basismaschinen oder Vorgaben zur Systemgestaltung, wie z. B. der Wahl der Service-Koordination, konzipiert. Aus diesem Grund erfolgt die Beschreibung von Entwurfsentscheidungen, die nicht vollständig formalisierbar sind, in Form von Überarbeitungsschritten der Konsolidierung. Die Durchführung der Systementwicklung wird zudem durch die methodischen Hilfsmittel *Architekturmodell*, *Vorgehensmodell* und die *Modellierungsansätze* auf den Modellebenen der Methodik unterstützt.

Aus dem Formalziel dieser Arbeit, die konsistente Abstimmung und Nachvollziehbarkeit zwischen den Modellebenen zu fördern, ergibt sich für die nachfolgende Konzeption der Entwicklungsschritte die Forderung, dass die Begrifflichkeiten der fachlichen AwS-Spezifikation in der REST-Architektur möglichst zu bewahren sind (Abschnitt 1.2.2). Abhängig von der verfolgten Zielsetzung und den Rahmenbedingungen des Entwicklungsprojektes können Kon-

solidierungen und Transformationen flexibel an geänderte Anforderungen angepasst werden, um dadurch den Automatisierungsgrad der Systementwicklungsaufgabe zu erhöhen bzw. zu verringern.

Die Vorstellung der sechs Entwicklungsschritte (T1-K3) ist nach folgendem Schema strukturiert:

- Überblicksartige Beschreibung des Entwicklungsschrittes anhand seiner Zielsetzung, Input (eingehende Artefakte), Output (erstellte Artefakte) und Lösungsansatz in tabellarischer Form.
- Vorstellung des Entwicklungsschrittes anhand eingesetzter Beziehungsmetamodelle und Ableitungsregeln (Transformation) bzw. Überarbeitungsschritten (Konsolidierung).
- Veranschaulichung der Durchführung des Entwicklungsschrittes anhand der erläuternden Fallstudie (Abschnitt 4.3).

Eine weiterführende Diskussion der vorgeschlagenen Entwicklungsschritte sowie von Alternativen zu den in dieser Arbeit getroffenen Entwurfsentscheidungen findet sich in Anhang B.2.

5.5.1 Erstellung des SOM-Geschäftsprozessmodells

Die oberste Ebene des Architekturmodells der SOM-R-Methodik beschreibt das Modell des betrieblichen Objektsystems in Form von SOM-Geschäftsprozessmodellen. Die Modellbildung auf der obersten Modellebene ist nicht Teil des Untersuchungsproblems dieser Arbeit. Die Modelle bilden den Startpunkt für die Durchführung der modellbasierten Systementwicklungsaufgabe (Abschnitt 1.2).

Für die SOM-Geschäftsprozessmodelle wird deshalb angenommen, dass sie einen *hinreichend feinen Detaillierungsgrad* besitzen. Alle relevanten Modellbestandteile zur Koordination und Durchführung der betrieblichen Leistungserstellung wurden mittels Objekt- und Transaktionszerlegung im Rahmen der Modellbildung dieser Ebene aufgedeckt und sind erfasst (Abschnitt 4.1.2).

Als weitere Annahme wird festgelegt, dass mit der Entwicklung von RESTful SOA eine *vollständige Ausschöpfung des Automatisierungspotenzials* des Geschäftsprozesses angestrebt wird. Dies bedeutet, dass die informationsverarbeitenden Aufgaben im SOM-GPM vollständig durch die Neuentwicklung eines AwS unterstützt werden sollen (Abschnitt 1.2.1).

Im SOM-Geschäftsprozessmodell werden die fachlichen Anforderungen an das zu entwickelnde AwS auf der Aufgabenebene aus struktur- und verhaltensorientierter Sicht beschrieben. Die Modellbildung dieser Ebene korrespondiert mit der Aktivität *fachliche Anforderungsdefinition (A-Ebene)* des SOM-R-Vorgehensmodells (Abschnitt 5.4).

5.5.2 Erste Transformation (T1): Ableitung der fachlichen AwS-Spezifikation

Die Ableitung der initialen fachlichen AwS-Spezifikation (Aufgabenträgerebene) erfolgt nach den Regeln der SOM-Methodik [FeSi13, S. 226–231]. Die Bausteine des SOM-Geschäftsprozessmodells (Quellsprache) und der fachlichen AwS-Spezifikation (Zielsprache) werden durch die

Spezifikation von Abbildungsbeziehungen auf der Metamodell-Ebene miteinander verknüpft und beschreiben eine *metamodellbasierte Transformation*. Der Entwicklungsschritt T1 wird in Tabelle 5.1 zusammenfassend dargestellt.

Zielsetzung	Initiale Ableitung der fachlichen Spezifikation des objektorientierten AwS (T1)
Input	IAS und VES des SOM-Geschäftsprozessmodells
Output	Initiales KOS, initiales VOS
Lösungsansatz	Metamodellbasierte Schema-Transformation (automatisiert). Spezifikation von Beziehungsmetamodell und Ableitungsregeln als methodische Hilfsmittel.

Tabelle 5.1: Zusammenfassung von Entwicklungsschritt *Transformation T1*

Die Abbildungsbeziehungen definieren ein **Beziehungsmetamodell**, welches Abbildung 5.11 darstellt [FeSi13, S. 235]. Das Ergebnis der Transformation T1 sind das initiale KOS und initiale VOS. Die Ableitung und der Aufbau von initialem KOS und VOS werden in der SOM-Methodik durch *Ableitungsregeln* konkretisiert. Grundlage für die Spezifikation des KOS sind IAS und VES des SOM-GPM. Ausgehend vom VES wird das VOS erstellt. Wie bereits erläutert, wird in dieser Arbeit die zeitkontinuierliche Parametrisierung nicht betrachtet (Abschnitt 4.1.2). Den Bezug zum Beziehungsmetamodell visualisieren die *Regelnummern* (Abbildung 5.11).

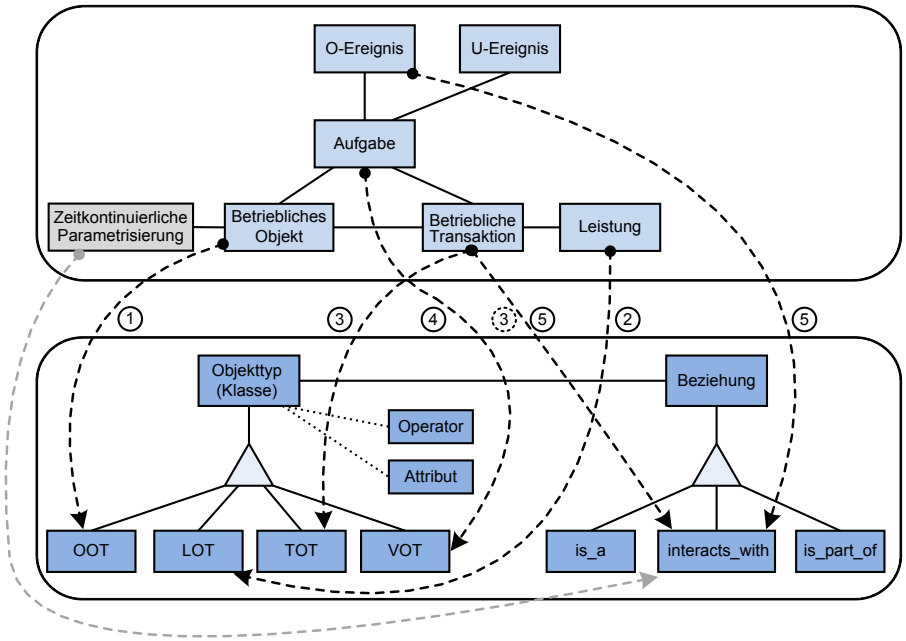


Abbildung 5.11: Beziehungsmetamodell des Entwicklungsschrittes *Transformation T1* (in Anlehnung an [FeSi13, S. 235])

Die erste Transformation T1 des Entwicklungsvorgehens entspricht der Ableitung der AwS-Spezifikation in der SOM-Methodik nach FERSTL und SINZ [FeSi13, S. 223 ff.] und wird deshalb nur in knapper Form erläutert.

Die **Ableitungsregeln des initialen KOS** sind (Ausführungen nach [FeSi13, S. 226]):

1. Betriebliche Objekte des IAS werden in objektspezifische Objekttypen (OOT) überführt. OOTs sind existenzunabhängig und werden in der linken Spalte des KOS angeordnet.

2. Leistungsspezifikationen des Geschäftsprozessmodells werden in existenzunabhängige, leistungsspezifische Objekttypen (LOT) transformiert und in der linken Spalte des KOS angeordnet.

3. Transaktionsspezifische Objekttypen (TOT) werden aus betrieblichen Transaktionen abgeleitet. Jeder TOT ist existenzabhängig von den zwei betrieblichen Objekten, die er verbindet sowie einer zugeordneten Leistung. Er wird mit diesen durch eine interacts_with-Beziehung verknüpft und im KOS rechts zu den Elementen angeordnet, von denen er abhängt (gestrichelt visualisiert).
   Zudem werden Existenzabhängigkeiten zu anderen TOTs analysiert und bei der Anordnung im KOS berücksichtigt. Abhängigkeiten ergeben sich aus der Ausführungsreihenfolge von Transaktionen im VES.
   Die Existenzabhängigkeiten zu betrieblichen Objekten und Leistungen können *transitiv* über eine Menge von Objekttypen spezifiziert werden.

Die **Ableitungsregeln der initialen VOS** lauten (Ausführungen nach [FeSi13, S. 231]):

4. Jede Aufgabe des VES wird in einen Vorgangsobjekttyp (VOT) überführt.

5. Die Ereignisbeziehungen zwischen Aufgaben werden in jeweils eine interacts_with-Beziehung zwischen VOTs abgebildet. Auch Transaktionen stellen Ereignisbeziehungen dar.

Zu jedem betrieblichen Objekt wird ein VOS erstellt. Entsprechend der Zuordnung von Aufgaben zu betrieblichen Objekten im VES werden die VOTs dem jeweiligen VOS zugeordnet.

BEISPIEL FLUGBUCHUNG (INITIALES KOS UND INITIALES VOS)

Die Durchführung des ersten Transformationsschrittes wird exemplarisch für das finale SOM-Geschäftsprozessmodell der eingeführten Fallstudie gezeigt (Abschnitt 4.3). Das Transformationsergebnis bilden die initialen Schemata KOS und VOS, welche Abbildung 5.12 jeweils ausschnittsweise darstellt. Die vollständigen Objektschemata finden sich in Anhang A.2.

Das initiale KOS besteht aus einer Menge von KOTs und den Beziehungen zwischen diesen. Die Leistung *Flugticketverkauf* und die betrieblichen Objekte des Geschäftsprozesses werden als existenzunabhängige LOTs bzw. OOTs links angeordnet. Die Transaktionen werden in TOTs überführt und nach ihren Abhängigkeiten zu betrieblichen Objekten, der unterstützten Leistung sowie der Ausführungsreihenfolge der Transaktionen im korrespondierenden VES mit den entsprechenden KOTs angeordnet. Beispielsweise koordiniert der KOT *Reservierung* die Vereinbarung des Flugticketverkaufs zwischen *Vertrieb* und *Kunde*. Seine Existenz ist abhängig von den KOTs *Buchung* und *Zahlungsdaten*, die entsprechend ihrer parallelen Ausführungsreihenfolge (Quelle VES) im KOS modelliert sind. Die Existenzabhängigkeiten zu den OOTs und dem LOT sind transitiv erfasst.

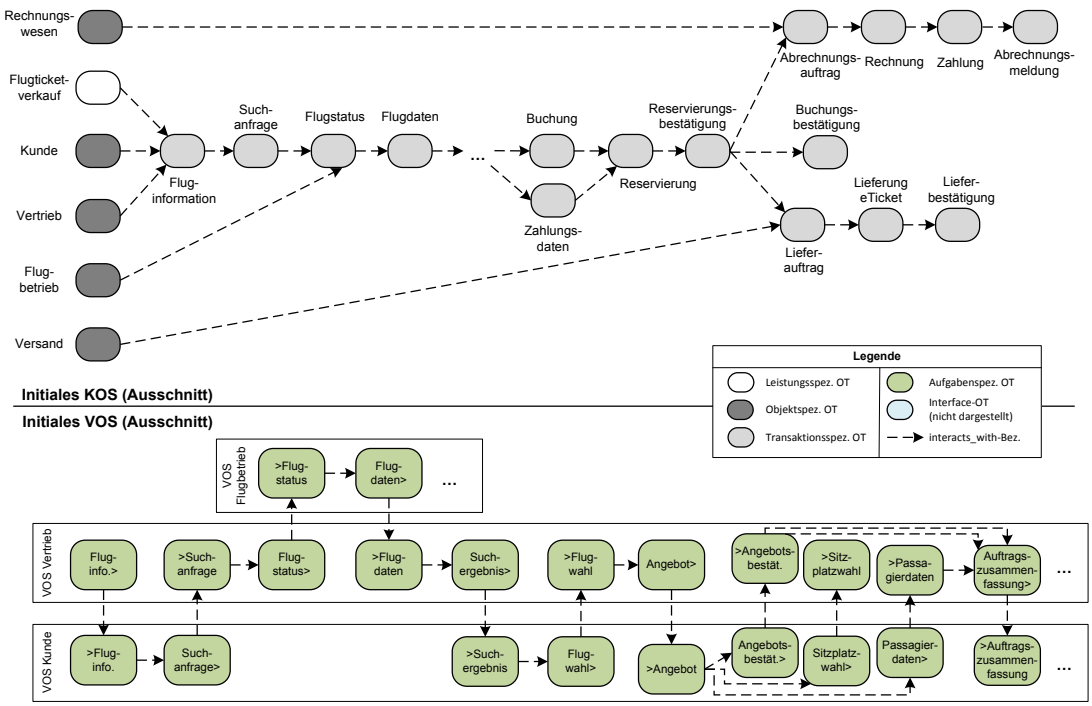


Abbildung 5.12: Initiales KOS und initiales VOS der Fallstudie (Ausschnitt)

Die VOTs des initialen VOS werden aus den Aufgaben des VES abgeleitet und nach ihren Abhängigkeiten zu betrieblichen Objekten dem jeweils zugehörigen VOS zugeordnet. Das betriebliche Objekt *Flugbetrieb* wird z. B. in ein VOS überführt, das die VOTs zu den Aufgaben *>Flugstatus* und *Flugdaten>* enthält. Die Ereignisbeziehungen und Transaktionen zwischen Aufgaben werden in Interaktionsbeziehungen zwischen den VOTs überführt.

5.5.3 Konsolidierung der fachlichen AwS-Spezifikation (K1)

Die initial abgeleiteten Objektschemata KOS und VOS werden in der Konsolidierung K1 in zwei Schritten überarbeitet [FeSi13, S. 227 und S. 231]. Im ersten Schritt (*Teilschritt K1.1*) wird das initiale KOS um objektorientierte Eigenschaften angereichert und strukturell überarbeitet. Das Ergebnis der Konsolidierung ist das konsolidierte KOS. Im zweiten Schritt (*Teilschritt K1.2*) wird die Spezifikation des initialen VOS detailliert. Im Zuge der VOS-Konsolidierung werden die Interfaceobjekttypen des IOS auf fachlicher AT-Ebene identifiziert. IOS und VOS werden in dieser Arbeit in einem gemeinsamen Schema dargestellt.

Zielsetzung	Konsolidierung der fachlichen Spezifikation des objektorientierten AwS (K1)
Input	Initiales KOS, initiales VOS
Output	K1.1: Konsolidiertes KOS K1.2: Konsolidiertes VOS (integrierte Darstellung von IOS und VOS)
Lösungsansatz	Methodisch geleitete Überarbeitung der Schemata (nicht-automatisiert). Spezifikation von Überarbeitungsschritten.

Tabelle 5.2: Zusammenfassung von Entwicklungsschritt *Konsolidierung K1*

Ergebnis der Konsolidierung sind die (konsolidierten) Schemata von KOS und VOS/IOS der überarbeiteten fachlichen AwS-Spezifikation. Die Durchführung des Entwicklungsschrittes entspricht der Aktivität *fachliche Anforderungsdefinition (AT-Ebene)* im Vorgehensmodell der

Methodik (Abschnitt 5.4). Die Konzeption des Entwicklungsschrittes wird in Anhang B.2 tiefergehend diskutiert.

5.5.3.1 Spezifikation des konzeptuellen Objektschemas (K1.1)

Die Konsolidierung des initialen KOS zielt auf die *fachliche Spezifikation des Anwendungssystems* aus strukturorientierter Sicht sowie der Definition der *Nachrichten*, die im Rahmen einer transaktionsorientierten Koordination der Leistungserstellung ausgetauscht werden. Das Ergebnis der Überarbeitung ist das *konsolidierte KOS,* welches die fachlichen Anforderungen an die Realisierung der domänenspezifischen Anwendungsfunktionalität und Datenhaltung in objektorientierter Form beschreibt (Abschnitt 4.1.3). Als Zwischenergebnis der Konsolidierung wird zudem eine *dokumentenorientierte Version des KOS* ausgewiesen, welche die Struktur der Nachrichten darstellt, die im Zuge der transaktionsorientierten Koordination der Leistungserstellung ausgetauscht werden.

Die **Überarbeitungsschritte zur Konsolidierung des initialen KOS (K1.1)** sind (Ausführungen nach [FeSi13, S. 227])[37]:

1. Entfernen von KOTs, die aus nicht-automatisierten Aufgaben oder Transaktionen abgeleitet werden.
2. Festlegen der Kardinalitäten der interacts_with-Beziehungen.
3. Zuordnen und spezifizieren der Attribute von KOTs.
   Auf Grundlage der zugewiesenen Attribute werden strukturelle Entwurfsentscheidungen bezüglich Normalisierung und Generalisierung getroffen:
   3.1 Komplexe Objekttypen, d. h. Objekttypen, die aus Datensicht mit nicht-normalisierten Diskurs- und Umweltobjekten sowie Informationsflüssen korrespondieren, werden unter Verwendung von *is_part_of-Beziehungen* aufgelöst.
   3.2 Die Generalisierung von Objekttypen, die mit Generalisierungen von betrieblichen Objekten sowie Flüssen korrespondieren, wird durch is_a-Beziehungen realisiert.
4. Bestimmen der Persistenzeigenschaften (persistent/transient) von KOTs.
   Während leistungs- und objektspezifische Objekttypen im Allgemeinen Hinweise auf zu persistierende Stammdaten geben, bilden transaktionsspezifische Objekttypen Ausführungszustände von Geschäftsvorfällen ab. Sie sind Ausgangspunkt für die Identifikation von persistenten oder transienten Transaktionsdaten [WoBe13, S. 1234 f.].

   Der Stand des KOS nach Schritt 4. markiert das *dokumentenorientierte KOS* als Zwischenergebnis der Konsolidierung. Dessen KOTs spezifizieren die Struktur derjenigen persistenten und transienten Daten im AwS, die aus dem SOM-GPM abgeleitet werden können.
5. Zuordnen und spezifizieren der Operatoren und Nachrichtendefinitionen von KOTs.
   Jedem KOT werden zunächst elementare Operatoren (z. B. CRUD-Operationen) sowie

[37] Die Konsolidierung des KOS folgt den Regeln der SOM-Methodik [FeSi13, S. 227]. Diese werden in dieser Arbeit um die Überarbeitungsschritte 4 und 6.2 erweitert, welche die Spezifikation der Struktur der bei der Abwicklung eines Geschäftsvorfalls ausgetauschten Nachrichten explizieren.

die Definitionen der ausgetauschten Nachrichten zugewiesen. Die Spezifikation weiterer Operatoren erfolgt im Zuge der Überarbeitung von VOTs.

6. Zusammenfassen von KOTs. Dies erfolgt auf Grundlage zweier, sich überlagernder Entscheidungskriterien:

   6.1 KOTs, die sich bezüglich ihrer Attribute und/oder Operatoren weitgehend überlappen, werden zusammengefasst.

   6.2 Persistente transaktionsspezifische Objekttypen, die einen *Ausführungszustand im Geschäftsablauf* beschreiben, werden zu einem oder mehreren persistenten Objekttypen zusammengefasst (fachliches Kriterium).
   Die Differenzierung des transaktionalen Ausführungszustands erfolgt durch Zuordnung eines neuen Attributs, das einen Wertebereich entsprechend der *Benennung der zusammengefassten Objekttypen* besitzt [WoBe13, S. 1234 f.].

Hinweis: Schritt 6. dient der Vermeidung von Daten- und Funktionsredundanz.

Das Ergebnis des vorgestellten Entwicklungsschrittes K1.1 ist das *konsolidierte KOS*. Je nach Zielsetzung der Überarbeitung wird die Entscheidung bezüglich einer KOT-Zusammenfassung durch das technische (überlappende Eigenschaften) oder das fachlich geprägte Kriterium dominiert. Der Schritte der KOS-Überarbeitung können mehrmals durchlaufen werden [FeSi13, S. 228].

5.5.3.2 Spezifikation des Vorgangsobjektschemas (K1.2)

Ziel der Überarbeitung des initialen VOS ist die *fachliche Spezifikation der Aufgabenträgerebene* des Informationssystems aus verhaltensorientierter Sicht [FeSi13, S. 230 f.]. Der Untersuchungsgegenstand ist dabei auf VOTs derjenigen VOS beschränkt, die mit *Aufgaben von Diskurswelttobjekten* korrespondieren. *Umwelttobjekte* kapseln ihre Vorgänge und werden im Rahmen der Systementwicklung nur aus der Außensicht betrachtet. Die VOS der Umwelttobjekte werden deshalb auf Aufgabenträgerebene auch als Black-Box dargestellt.

Die **Überarbeitungsschritte für das initiale VOS (K 1.2)** sind (nach [FeSi13, S. 231])[38]:

1. Entfernen von VOTs, die aus nicht-automatisierten Aufgaben abgeleitet werden.
2. Zuordnen von Attributen zu VOTs. Die Attribute werden in Form eines Teilgraphen des überarbeiteten KOS spezifiziert. Die zugeordneten konzeptuellen Objekttypen beschreiben die Objektstruktur, auf die während der Vorgangsdurchführung zugegriffen wird. Im Allgemeinen überlappen sich die Teilgraphen von unterschiedlichen VOTs.
3. Definieren der Nachrichten, die von VOTs gesendet oder empfangen werden, und Bestimmung ihres Automatisierungsgrads.
   Für Nachrichten, die zwischen VOTs unterschiedlicher VOS ausgetauscht werden, ist zu

[38] Die Überarbeitung des VOS folgt den Schritten der SOM-Methodik [FeSi13, S. 231]. Zur Vorbereitung der Identifikation von IOTs werden in der VOS-Konsolidierung der SOM-R-Methodik zusätzlich die Automatisierungsgrade der Nachrichten entsprechend den Automatisierungsgraden von Transaktionen bestimmt (K1.2, Schritt 3). Die Definition der gesendeten/empfangenen Nachrichten der VOTs erfolgt auf Basis der TOTs des dokumentenorientierten KOS.

berücksichtigen, dass sie in einer Abhängigkeitsbeziehung zur Strukturbeschreibung derjenigen TOTs des dokumentenorientierten KOS stehen, die aus den korrespondierenden Transaktionen des VES abgeleitet wurden.

4. Spezifizieren der Operatoren von VOTs.
   Die Lösungsverfahren der Operatoren werden als logische Navigation durch einen Teilgraphen des KOS beschrieben.
5. Zusammenfassen derjenigen VOTs, deren Aufgaben aus Gründen der Wahrung der semantischen Integrität des AwS immer gemeinsam durchgeführt werden.

Die Schritte zur Überarbeitung des VOS können in mehreren Zyklen durchlaufen werden. Hierbei findet stets eine Abstimmung mit den Ergebnissen der KOS-Konsolidierung statt.

Die Konsolidierungsschritte des VOS werden in dieser Arbeit um die Schritte zur **Identifikation und Spezifikation der Interface-Objekttypen** auf fachlicher AT-Ebene erweitert:

6. Zuordnen von Interface-Objekttypen zu Vorgangsobjekttypen des VOS.
   6.1 Zuordnen von IOTs zur Spezifikation von *Kommunikationskanälen mit der Umwelt*.
   Jedem VOT eines VOS der Diskurswelt (abgeleitet aus Diskursweltobjekt), der mit dem VOT eines VOS der Umwelt (abgeleitet aus Umweltobjekt) in Beziehung steht, wird ein IOT zugeordnet.
   6.2 Zuordnen von IOTs zur Spezifikation von *Kommunikationskanälen zwischen unterschiedlichen Aufgabenträgern der Diskurswelt*.
   VOTs die innerhalb der Diskurswelt in Kommunikationsbeziehung mit VOTs stehen, die durch andersartige AT (hier: AwS) realisiert sind, wird ein IOT zugeordnet.
7. Bestimmen der Schnittstelle von IOTs anhand der Merkmale der beteiligten Aufgabenträgertypen (CCK, MCK), der Kommunikationsform (synchron, asynchron) sowie der Zuständigkeit für den Empfang (E) und/oder das Senden (S) von Nachrichten.
8. Spezifizieren der Eigenschaften von IOTs durch Zuordnung von Attributen, Nachrichten und Operatoren.
9. Aktualisieren der Operatoren von VOTs, um die Aufrufe von Methoden der zugeordneten IOTs zu realisieren.

Als Grundlage für die IOS-Konsolidierungsschritte dienen die Ausführungen bezüglich der Zuordnung von IOTs zu VOTs nach [Mali97, S. 59 f.], die in Schritt 6 überarbeitet und um die Schritte 7 bis 9 zur Spezifikation der Schnittstelle und Eigenschaften erweitert werden. Hinweise auf die Festlegung der Schnittstelleneigenschaften geben die Beziehungen sowie die Automatisierungsgrade der Nachrichten im VOS ([FeSi13, S. 234], [Mali97, S. 59 f.]). Genauere Anforderungen an die Wahl des einzusetzenden Kommunikationsprotokolls und der konkreten Ausgestaltung der Schnittstellenbeziehung sind von der eingesetzten Technologie abhängig.

Ergebnis der Konsolidierung K1.2 ist das *konsolidierte VOS*, das die fachliche Spezifikation der Vorgänge und Kommunikationskanäle eines Anwendungssystems als Menge von VOTs und IOTs und deren Beziehungen beschreibt.

BEISPIEL FLUGBUCHUNG (KONSOLIDIERTES KOS UND VOS)

Die Schritte der Konsolidierung der fachlichen AwS-Spezifikation werden nun am Beispiel der Fallstudie erläutert. Das konsolidierte KOS und einen Ausschnitt des konsolidierten VOS der Fallstudie zeigt Abbildung 5.13.

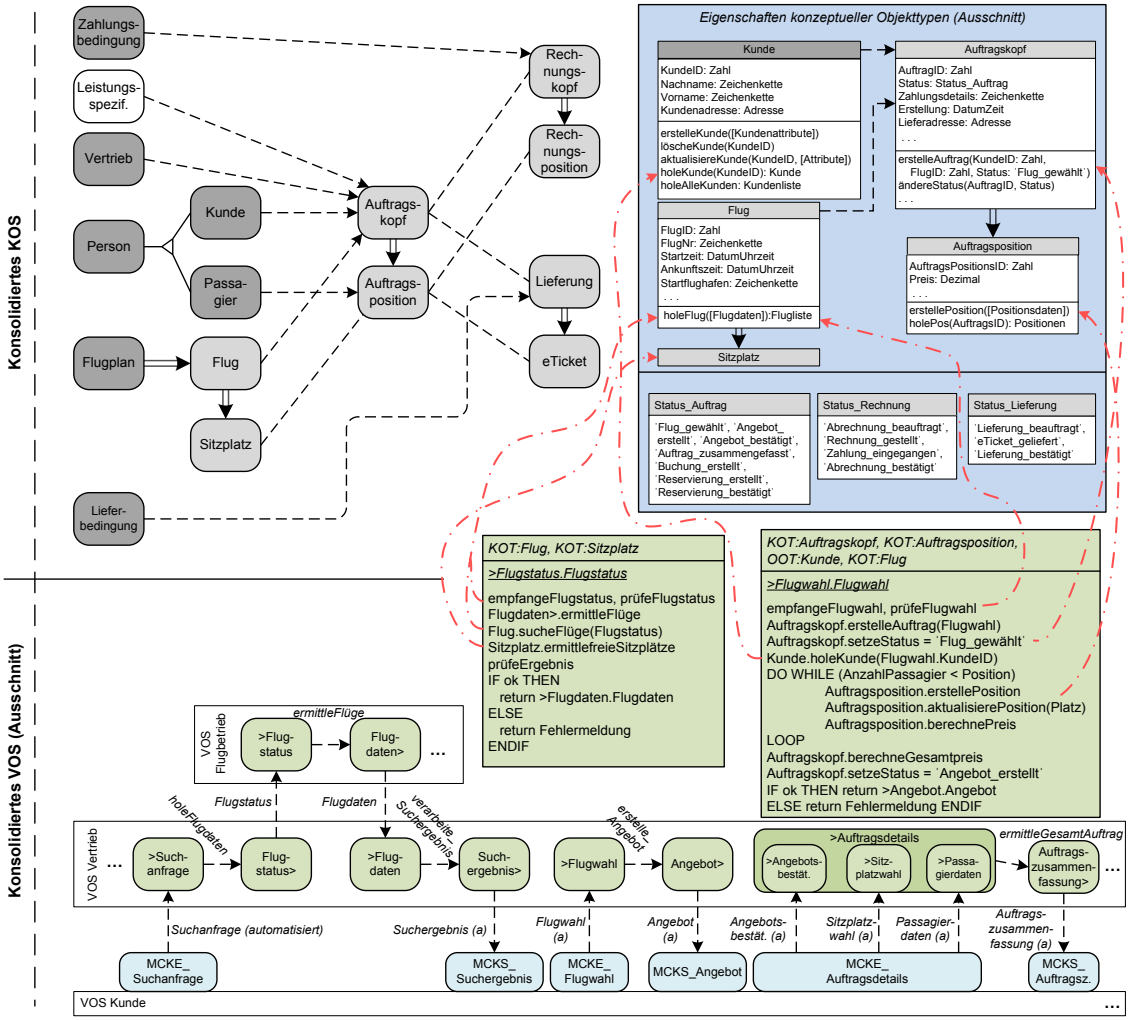


Abbildung 5.13: Konsolidiertes KOS und Ausschnitt des konsolidierten VOS der Fallstudie

Im ersten Schritt der KOS-Konsolidierung werden keine Objekttypen des KOS entfernt, da eine vollständige Ausschöpfung des Automatisierungsgrades im Geschäftsprozess angestrebt wird (K1.1, Schritt 1). Es erfolgt eine Überarbeitung der Kardinalitäten der bestehenden Beziehungen und die Zuordnung von Attributen zu Objekttypen (K1.1, 2 und 3).

Auf dieser Basis werden die TOTs *Flugwahl*, *Angebot*, *Angebotsbestätigung*, *Auftragszusammenfassung*, *Buchung*, *Reservierung* sowie deren *Bestätigungen* normalisiert und in einen *Kopf* und die zugehörigen *Positionen* zerlegt. Die Positionen referenzieren die einzelnen Sitzplätze des Fluges und später auch Passagierdaten, die im Zuge der Vereinbarung zwischen Vertrieb und Kunde bestimmt werden. Selbiges gilt für die TOTs *Abrechnung* und *Lieferung*. Der TOT *Flugdaten* wird in die Objekttypen *Flug* und *Sitzplatz* zerlegt (K1.1, 3.1). Durch die Generalisierung von *Kunde* und *Passagier* (TOT Passagierdaten) entsteht der Objekttyp *Person* (K1.1, 3.2). Die OOTs *Rechnungswesen* und *Versand* spezifizieren die Bedingungen und Optionen, die für eine Durchführung der Abrechnung und Lieferung gelten. Beide OOTs werden deshalb im

Zuge der Festlegung von Eigenschaften in *Zahlungs*- und *Lieferbedingung* umbenannt. Aufgabe des *Flugbetriebes* ist die Verwaltung der Flugpläne. Dementsprechend wird auch seine Benennung in *Flugplan* geändert.

Im Schritt zur Bestimmung der Persistenzeigenschaften (K1.1, 4) werden *Fluginformation*, *Suche*, *Flugstatus* und *Suchergebnis* als transiente Objekttypen ausgewählt, da die versendeten Fluginformationen und die empfangenen Suchanfragen im AwS nicht dauerhaft gespeichert werden. Sie dienen im weiteren Verlauf jedoch als Input für die Ableitung von Dokumenttypen (T2). Nach Zuordnung elementarer Manipulationsoperatoren und Nachrichtendefinitionen (K1.1, 5) werden Objekttypen zusammengefasst, um mögliche Funktions- und Datenredundanzen zu reduzieren (K1.1, 6).

Eine OT-Zusammenfassung erfolgt für die Teil-Ganzes-Beziehungen (Kopf-zu-Position) von *Flugwahl*, *Angebot*, *Angebotsbestätigung*, *Auftragszusammenfassung*, *Buchung*, *Reservierung* sowie deren *Bestätigungen* zu den Objekttypen *Auftragskopf* und *-position*. Zur Beschreibung des Ausführungszustands eines Auftrags wird ein *Status-Attribut* definiert, dessen Wertebereich die Bezeichnungen zusammengefasster Objekttypen enthält (z. B. Flug gewählt, Angebot erstellt, usw.). Entsprechend diesem Vorgehen werden die Objekttypen von *Lieferung* und *Abrechnung* überarbeitet. Der TOT *Zahlungsdaten* geht als Attribut in den *Auftragskopf* ein.

Die Konsolidierung des VOS sieht zunächst das Entfernen derjenigen Modellbausteine vor, die nicht automatisiert werden (entfällt in der Fallstudie). Anschließend werden den VOTs Attribute, Operatoren und Nachrichtendefinitionen zugeordnet (K1.2, Schritte 1-3). Die Interaktion zwischen AwS und Kunde wird automatisiert (a) durchgeführt. Die Benennung der ausgetauschten Nachrichten zwischen den VOTs unterschiedlicher VOS orientiert sich im vorliegenden Beispiel an der Benennung der entsprechenden Transaktionen im SOM-Geschäftsprozessmodell, und damit den TOTs des initialen KOS. So löst z. B. der Empfang der automatisiert übertragenen Nachricht *Flugwahl (a)* die Durchführung des Operators *empfangeFlugwahl* von *VOT >Flugwahl* aus und startet dadurch den Workflow der Aufgabe. Die Lösungsverfahren der VOT-Operatoren werden als logische Navigation über den zugeordneten Ausschnitt des KOS (VOT-Attribute) und den Zugriff auf die Operationen der Objekttypen in textueller Form spezifiziert (K1.2, 4). Für die Beschreibung der VOT-Lösungsverfahren ist im Beispiel keine Sprachdefinition vorgegeben (vgl. [FeSi13, S. 233]). Die Navigation durch einen Ausschnitt des KOS wird am Beispiel des Empfangs der Suchdaten (Suchanfrage) durch VOT *>Flugdaten* und für den gewählten Flug durch VOT *>Flugwahl* dargestellt. Die VOTs des VOS *Kunde* werden nicht explizit modelliert, da diese die externen personellen Nutzer des Systems repräsentieren. Ein Zusammenlegen von Objekttypen erfolgt für *>Angebotsbestätigung*, *>Sitzplatzwahl* und *>Passagierdaten* zu einem VOT *>Auftragsdetails* (K1.2, 5). Die ausgetauschten Nachrichten (Angebotsbestätigung, Sitzplatzwahl, Passagierdaten) werden getrennt dargestellt, um weitergehende Überarbeitungen zu unterstützen.

Die Erstellung und Konsolidierung des IOS beginnt mit der Zuordnung von IOTs zu denjenigen VOTs, die mit dem Kunden (Umweltobjekt) kommunizieren (K1.2, 6). Anschließend werden die Schnittstellenmerkmale und weitere IOT-Eigenschaften spezifiziert sowie eine Anpassung der VOTs geprüft (K1.2, 7-8). Beispielsweise dient der IOT *MCKS_Angebot* dem Senden (S) von Angeboten an einen menschlichen Nutzer (MCK). Hierzu kapselt er eine Menge an Operatoren,

welche die Formatierung des Angebots in eine menschenlesbare Form (z. B. HTML) und dem Senden der Nachricht (z. B. per HTTP an Web-Browser oder als E-Mail) realisieren. Eine konkrete Ausgestaltung des IOT erfolgt auf der softwaretechnischen Ebene.

5.5.4 Zweite Transformation (T2): Ableitung des REST-Architekturmodells

Das Ziel des Entwicklungsschrittes *Transformation T2* ist die initiale Ableitung der Modellebene der Softwarearchitektur der RESTful SOA (REST-Architekturmodell). Die Transformation T2 wird in den zwei *Teilschritten T2.1* und *T2.2* durchgeführt. Den Input bilden die konsolidierten Schemata von KOS und VOS, die in T2.1 zunächst in die initialen Schemata OS-ED und OS-VD transformiert werden, welche die Dienste und Innensicht der RESTful SOA in initialer Form spezifizieren. Anschließend erfolgt die Ableitung einer initialen Beschreibung des bereitgestellten Service-Vertrags in Form von Ressourcenschema sowie der, im Rahmen der Service-Nutzung, ausgetauschten Dokumente (Dokumentenschema) in Teilschritt T2.2.

Zielsetzung	Initiale Ableitung der Softwarearchitekturspezifikation der RESTful SOA (T2)
Input	Konsolidiertes KOS, konsolidiertes VOS
Output	T2.1: Initiales OS-ED, initiales OS-VD; T2.2: Initiales Ressourcenschema, initiales Dokumentenschema
Lösungsansatz	Metamodellbasierte Schema-Transformation (automatisiert). Spezifikation von Beziehungsmetamodell und Ableitungsregeln als methodische Hilfsmittel.

Tabelle 5.3: Zusammenfassung von Entwicklungsschritt *Transformation T2*

Die Durchführung des Entwicklungsschrittes T2 entspricht den Aktivitäten *Service-Identifikation* und *grober Serviceentwurf* im Vorgehensmodell der Methodik (Abschnitt 5.4). Die Konzeption der Entwurfsschritte von T2 wird in Anhang B.2.2 diskutiert.

5.5.4.1 Initiale Ableitung von OS-ED und OS-VD (T2.1)

Das **Beziehungsmetamodell des Teilschrittes T2.1** wird zur besseren Übersichtlichkeit in tabellarischer Form beschrieben. Tabelle 5.4 und Tabelle 5.5 erfassen die *Abbildungsbeziehungen* zwischen den Metamodellbausteinen der Modellebenen.

Konsolidiertes KOS	**Initiales OS-ED**
Konzeptueller Objekttyp (persistent) (KOT)	Entitätsspezifischer Objekttyp (EOT)
Eigenschaften (Attribute, Operatoren) KOT	Eigenschaften (Attribute, Methoden[39]) EOT
Beziehungen zwischen zwei KOTs	Beziehungen zwischen zwei EOTs

Tabelle 5.4: Abbildungsbeziehungen zwischen konsolidiertem KOS und initialem OS-ED

[39] Die Operatoren eines Objekttyps im REST-Architekturmodell werden im Folgenden als „Methoden“ bezeichnet.

Konsolidiertes VOS	Initiales OS-VD
Vorgangsobjekttyp (VOT)	Elementarer Vorgangsobjekttyp (eVOT)
Eigenschaften (Attribute, Operatoren) VOT	Eigenschaften (Attribute, Methoden) eVOT
Beziehung zwischen zwei VOTs	Beziehung zwischen zwei eVOTs
Interface-Objekttyp (IOT)	Interface-Objekttyp (REST-Architektur) (R-IOT)
Eigenschaften (Attribute, Operatoren) IOT	Eigenschaften (Attribute, Methoden) R-IOT
Beziehung zwischen IOT und VOT	Beziehung zwischen R-IOT und eVOT

Tabelle 5.5: Abbildungsbeziehungen zwischen konsolidiertem VOS und initialem OS-VD

Die Transformation T2.1 umfasst Abbildungsbeziehungen und Ableitungsregeln. Die fachlichen Objekttypen sowie ihre Eigenschaften und Beziehungen werden in die Softwarebausteine der REST-Modellebene überführt. Das Vorgehen zur Realisierung der Abbildung zwischen den Metaobjekten von OS-ED und OS-VD leiten Ableitungsregeln an. Die **Ableitungsregeln des initialen OS-ED** lauten:

1. Jeder (persistente) KOT des KOS wird in einen entitätsspezifischen Objekttypen (EOT) des OS-ED überführt. Die EOTs werden entsprechend ihrer existenziellen Abhängigkeiten im KOS von links nach rechts angeordnet.
2. Eigenschaften von KOTs, wie Attribute und Operatoren einschließlich Nachrichtendefinitionen oder Persistenzeigenschaft, werden in Eigenschaften der abgeleiteten EOTs überführt.
3. Is_a-, is_part_of- und interacts_with-Beziehungen zwischen KOTs werden in äquivalente Beziehungstypen (sowie Kardinalitäten) zwischen EOTs transformiert.

Anschließend erfolgt die Überführung und Spezifikation der Vorgangssicht. Die **Ableitungsregeln des initialen OS-VD** lauten:

4. Für jedes VOS wird ein OS-VD im REST-Architekturmodell erstellt.
5. Jeder VOT eines VOS wird in einen elementaren Vorgangsobjekttypen (eVOT) des zugehörigen OS-VD überführt.
6. Die Eigenschaften von VOTs, wie z. B. Attribute, Operatoren und Nachrichtendefinitionen, werden in Eigenschaften der entsprechenden eVOTs überführt.
7. Die interacts_with-Beziehungen zwischen VOTs werden in äquivalente Beziehungstypen zwischen korrespondierenden eVOTs transformiert.
8. IOTs eines VOS werden in Interface-Objekttypen der REST-Architektur (R-IOT) des zugehörigen OS-VD überführt.
9. Die Eigenschaften von IOTs, wie Attribute, Operatoren und auch Schnittstellenmerkmale werden übernommen.
10. Beziehungen zwischen IOTs und VOTs werden in äquivalente Beziehungstypen zwischen R-IOTs und eVOTs transformiert.

Das Ergebnis der Transformation T2.1 sind das *initiale OS-ED* (Schritte 1-3) sowie die *initialen OS-VDs* (Schritte 4-9). Sie bilden den Ausgangspunkt für die Spezifikation der *Innensicht* der Softwarearchitektur der RESTful SOA. Das OS-ED beschreibt das REST-Architekturmodell aus *strukturorientierter Sicht*. Die *verhaltensorientierte Sicht* auf das REST-Architekturmodell sowie die Interaktionsbeziehungen zur Umwelt erfasst das OS-VD.
Die Transformation dient der konzeptuellen Trennung der initialen Version des REST-Architekturmodells von den konsolidierten Schemata der fachlichen Ebene. Zudem können, neben der Abbildung von Schemabausteinen, alle plattformspezifischen Anpassungen, die bereits automatisierbar sind, in dem Transformationsschritt gekapselt werden. Beispielsweise kann eine standardmäßige Typisierung von Attributen erfolgen.

5.5.4.2 Initiale Ableitung von Ressourcen- und Dokumentenschema (T2.2)

Aus der *Außenperspektive* stellt die RESTful SOA die Funktionalität ihrer Dienste als Menge von Ressourcen bereit. Der *initiale Service-Vertrag der RESTful SOA* wird durch ein Ressourcenschema und Dokumentenschema spezifiziert. Das *Dokumentenschema* definiert auf Basis des KOS die Dokumenttypen (Repräsentationen), die im Rahmen der Service-Nutzung über HTTP-Nachrichten ausgetauscht werden (vgl. Metamodell des Ressourcenschemas, Abschnitt 5.3.4).

ABLEITUNG DES INITIALEN RESSOURCENSCHEMAS

Ausgehend von den Objekttypen des konsolidierten KOS und VOS werden, korrespondierend zur Spezifikation der initialen EOTs und eVOTs, Entitätsressourcen (ER) und elementare Vorgangsressourcen (eVR) abgeleitet. Des Weiteren wird die einheitliche Schnittstelle jeder Ressource mit Standardwerten für ihren Identifikator und die veröffentlichten Standard-Operatoren angereichert.

Die Transformation von persistenten KOTs oder VOTs in Entitäts- bzw. elementare Vorgangsressourcen sowie die Bestimmung der Identifikatoren und Operatoren wird in einem Beziehungsmetamodell definiert. Dieses wird in Form von Abbildungsbeziehungen und Ableitungsregeln beschrieben. Identifikatoren werden in Form von URLs und die Operatoren in Form von HTTP-Methoden spezifiziert (Abschnitte 3.4.1 und 3.4.2). Die Konzeption der initialen Ableitung des Ressourcenschemas basiert auf den Ausführungen von WOLF [Wolf12, S. 1657].

Der Identifikation und initialen Spezifikation von Ressourcen aus den fachlichen Objekttypen (KOT (p), VOT) liegen folgende allgemeinen Transformationsregeln zugrunde:

1. Jeder *Objekttyp* wird in eine Individualressource und eine Listenressource transformiert (Ressourcentypen, Abschnitt 3.3.2.1).
2. Jeder *Listenressource* werden als Identifikator der Name des Objekttypen (URL: */ressource/*) sowie die HTTP-Operatoren `GET` und `POST` zugeordnet.
3. Jeder *Individualressource* werden als Identifikator der Name des Objekttypen und ein Platzhalter für die ID einer Ressourceninstanz (URL: */ressource/{id}*) sowie die HTTP-Operatoren `GET`, `DELETE` und `PUT` zugeordnet.

Nachfolgend werden in Tabelle 5.6 die **Abbildungsbeziehungen zur Ableitung des initialen Ressourcenschemas der Entitätsressourcen (ER)** aus dem konsolidierten KOS zusammengefasst.

Konsolidiertes KOS	Ressourcentyp (ER)	Identifikator (URL)	Operator (HTTP)
Konzeptueller Objekttyp	Listenressource	/ressource/	GET, POST
	Individualressource	/ressource/{id}	GET, PUT, DELETE
is_part_of-Beziehung	Liste der Subressourcen	/ressource/{id}/subressource/	GET, POST
	Subressource	/ressource/{id}/subressource/{id}	GET, PUT, DELETE
is_a-Beziehung	Liste der Subressourcen	/ressource/subressource/	GET, POST
	Subressource	/ressource/subressource/{id}	GET, PUT, DELETE

Tabelle 5.6: Abbildungsbeziehungen zwischen dem konsolidierten KOS und dem initialen Ressourcenschema (Entitätsressourcen) (in Anlehnung an [Wolf12, S. 1657])

Das Vorgehen zur Überführung von Beziehungen zwischen KOTs wird durch die **Ableitungsregeln der Beziehungstypen** konkretisiert:

- Die *is_part_of-Beziehung* (*Aggregation*) zwischen zwei KOTs wird im Identifikator als Beziehung einer Subressource (Teil) zur „aggregierenden" Individualressource (Ganzes) realisiert. Beispielsweise könnte die Aggregationsbeziehung von Auftragspositionen zu einem Auftragskopf als Subressource in Form einer Listenressource */auftrag/{id}/auftragsposition/* und Individualressource */auftrag/{id}/auftragsposition/{id}* dargestellt werden.
- Die Semantik der *is_a-Beziehung* (*Generalisierung*) zwischen KOTs wird als Subressourcenbeziehung zur „generalisierten" Listenressource definiert. Die Ressourcen des spezialisierten Objekttyps werden als Subressource unter den URL-Pfad des generalisierten Objekttypen eingeordnet. Beispielsweise ist Beschreibung der Generalisierung von Person zu Kunde durch die URL */person/kunde/* möglich.

Das vorgeschlagene Schema zur Abbildung von is_part_of- und is_a-Beziehungen zwischen KOTs in die URLs von Ressourcen markiert einen Standardfall von dem auch ggfs. abgewichen werden kann.

Die **Abbildungsbeziehungen zur Ableitung des initialen Ressourcenschemas (eVR)** aus den VOTs der konsolidierten VOS fasst Tabelle 5.7 zusammen:

Konsolidiertes VOS	Ressourcentyp (eVR)	Identifikator (URL)	Operator (HTTP)
Vorgangsobjekttyp	Listenressource	/ressource/	GET, POST
	Individualressource	/ressource/{id}	GET, PUT, DELETE

Tabelle 5.7: Abbildungsbeziehungen zwischen dem konsolidierten VOS und dem initialen Ressourcenschema (Vorgangsressourcen) (in Anlehnung an [Wolf12, S. 1657])

Jeder VOT korrespondiert mit einer Menge von *elementaren Vorgangsressourcen* eines *elementaren Vorgangsdienstes*. Die Transformation des konsolidierten VOS in ein initiales Res-

sourcenschema beschränkt sich auf die Überführung von VOTs in eVRs. Weitere Abbildungsbeziehungen werden nicht definiert, da im VOS lediglich interacts_with-Beziehungen zwischen VOTs bestehen. Die Identifikation und Spezifikation von nicht-elementaren Vorgangsressourcen (neVR) erfolgt im Rahmen der Konsolidierung des OS-VD (Abschnitt 5.5.5.2).

Die Interacts_with-Beziehungen (Assoziation) zwischen KOTs oder zwischen VOTs geben Hinweise auf die Realisierung von Verknüpfungen zwischen Ressourcen und dienen somit zur *Gestaltung der Hypermedia*. Die Definition einer standardmäßigen Abbildung in die URL-Struktur verknüpfter Ressourcen erfolgt nicht (siehe hierzu Abschnitt 5.5.5).

Für die Standard-Spezifikation von URL-Identifikator und HTTP-Operatoren der einheitlichen Ressourcenschnittstelle gelten in dieser Arbeit zwei *Regeln*:

- *Regel „Ableitung Identifikator“*: Der Aufbau von URLs orientiert sich grundsätzlich an den modellierten Existenzabhängigkeiten zwischen Objekttypen (z. B. is_a, is_part_of). Das Hilfsmittel zur Abbildung von Abhängigkeiten ist das Konzept der *Subressource* (Abschnitt 3.3.2.1).
  Als Beispiel hierfür sei der hierarchische Aufbau der Ressourcen-URL für Objekttypen genannt, die in einer is_part_of-Beziehung (Teil-Ganzes) stehen (*/ressource/{id}/subressource/*).
- *Regel „Ableitung HTTP-Operatoren“*: Die Operatoren `DELETE` und `PUT` werden zur Manipulation (Löschen und Ändern) von Instanzen verwendet und beziehen sich immer auf die ID einer Individualressource. `POST` dient dem Erstellen neuer Instanzen von Ressourcen, zu der es bisher keine ID gibt. Der „Create“-Operator wird deshalb nur für Listenressourcen definiert (Abschnitt 3.3.2.1).

Abschließend sei im Hinblick auf die Ableitung des initialen Ressourcenschemas noch einmal hervorgehoben, dass es sich bei den definierten Abbildungsbeziehungen/-regeln um ein standardisiertes Vorgehen handelt. Die Nachbildung der Semantik unterschiedlicher Beziehungstypen im Identifikator ist immer auch von externen Anforderungen an die Systementwicklungsaufgabe abhängig. Eine Anwendung der vorgestellten Regeln sollte deshalb stets reflektiert und bei Bedarf angepasst werden.

Ableitung des initialen Dokumentenschemas

Das Dokumentenschema spezifiziert die Struktur der *Ressourcen-Repräsentationen*, die im Rahmen der Nutzung von den Diensten der RESTful SOA gesendet und empfangen werden. Der Ausgangspunkt für die Ableitung des initialen Dokumentenschemas ist das konsolidierte KOS. Die Ableitung der Transaktionsdatendokumenttypen kann zudem unter Zuhilfenahme der Spezifikationen aus den Nachrichten der Interaktionsbeziehungen im VOS sowie dem dokumentenorientierten KOS als zusätzliche Strukturinformationen unterstützt werden (vgl. auch Diskussion in Anhang B.2.2). Als Basis für die Ableitung von Dokumenttypen schlagen Wolf und Benker die dokumentenorientierte Konsolidierung des KOS vor [WoBe13], die in K1.1, Schritt 4 aufgegriffen wurde.

Konsolidiertes KOS	Dokumenttyp
Objektspezifischer Objekttyp	Stammdatendokumenttyp
Leistungsspezifischer Objekttyp	Stammdatendokumenttyp
Transaktionsspezifischer Objekttyp	Transaktionsdatendokumenttyp
Objekttyp-Attribut	Dokumenttyp-Attribut
interacts_with-Beziehung	Dokumentenreferenz
is_part_of-Beziehung	Zusammenfassen von Dokumenttypen

Tabelle 5.8: Abbildungsbeziehungen zwischen dem konsolidierten KOS und dem initialen Dokumentenschema (in Anlehnung an [WoBe13, S. 1234 f.])

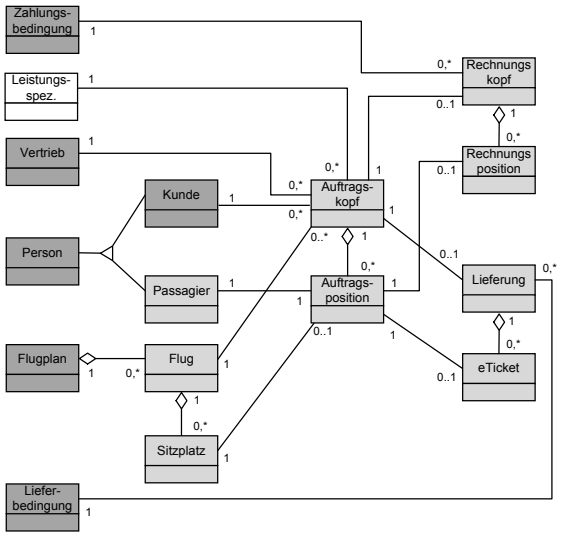
- *Stammdatendokumenttypen* (*SD-Dokumenttyp*) leiten sich aus den linksstehenden, existenzunabhängigen Objekttypen des konsolidierten KOS ab. OOTs und LOTs sind unabhängig von der Durchführung des Geschäftsprozessablaufs und beschreiben die Daten der am Geschäftsprozess beteiligten betrieblichen Objekte bzw. der Leistungsspezifikation. Anhand ihrer Persistenzeigenschaft (p/t) wird bestimmt, ob die Objekttypen dauerhaft zu speichern sind. Für jeweils einen dauerhaft zu speichernden Objekttypen wird ein korrespondierender *Stammdatendokumenttyp* abgeleitet, dem die Attribute des Objekttypen zugeordnet werden.
- *Transaktionsdokumenttypen* (TD-Dokumenttyp) bilden konkrete Ausführungszustände ab, die bei der Koordination und Durchführung der Leistungserstellung eines Geschäftsprozesses entstehen. Zur Ableitung der Dokumenttypen werden die (transienten und persistenten) TOTs des konsolidierten KOS in Transaktionsdatendokumenttypen überführt. Die Struktur der TD-Dokumente wird auf Basis der Attribute der jeweils zugehörigen Objekttypen spezifiziert. Zur Abbildung des *Ressourcenzustands* wird jedem TD-Dokumenttyp ein „Status-Attribut“ zugewiesen und zulässige Ausprägungen aus den korrespondierenden (zusammengefassten) Objekttypen des konsolidierten KOS übernommen. Jedes TD-Dokument ist hier als Nachricht interpretierbar, die während des Geschäftsprozessablaufs zur Koordination der Leistungserstellung zwischen den beteiligten betrieblichen Objekten ausgetauscht wird. Die übermittelten Dokumente werden im OS-VD erfasst.
- *Interaktionsbeziehungen* (interacts_with) zwischen Objekttypen werden in *Referenzen* (Links) überführt, die Dokumenttypen verknüpfen. Das existenzabhängige Dokument referenziert dabei das existenzunabhängige Dokument (Standardfall).
- Objekttypen, die durch eine *Aggregationsbeziehung* (is_part_of) verknüpft sind, werden in einem gemeinsamen Dokumenttyp beschrieben (Standardfall). Teil-Ganzes-Strukturen markieren existenzielle Abhängigkeiten, weshalb die Zustandsinformationen verknüpfter Komponenten normalerweise gemeinsam erfasst, übertragen und verarbeitet werden.

BEISPIEL FLUGBUCHUNG (INITIALE SCHEMATA DES REST-ARCHITEKTURMODELLS)

Die Schemata KOS und VOS/IOS der konsolidierten fachlichen AwS-Spezifikation werden mittels metamodellbasierter Schema-Transformation in die Schemata des initialen REST-Architekturmodells überführt. In der Fallstudie der vorliegenden Arbeit werden Repräsentationen

als JSON- (CCK) bzw. HTML-Dokumente (MCK) formatiert. Abbildung 5.14 stellt das Ergebnis der Transformation T2 für die Fallstudie dar (Abschnitte 5.5.4.1 und 5.5.4.2).

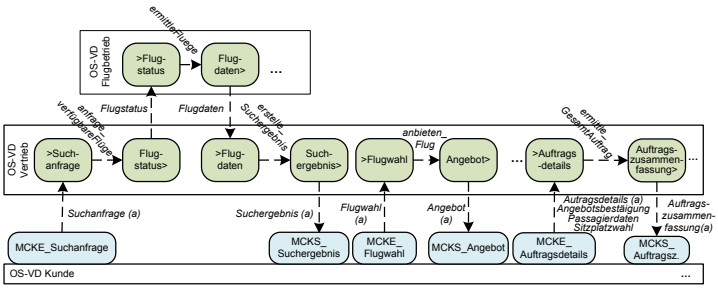
Initiales OS-ED (Klassendiagramm)



Initiales Ressourcenschema (ER)

Entitätsdienst	URL der Ressourcen	HTTP-Operatoren
Leistungsspezifikation	/leistungsspezifikation/	GET, POST
	/leistungsspezifikation/{id}	GET, PUT, DELETE
Vertrieb	/vertrieb/	GET, POST
	/vertrieb/{id}	GET, PUT, DELETE
Person	/person/	GET, POST
	/person/{id}	GET, PUT, DELETE
Kunde	/person/kunde/	GET, POST
	/person/kunde/{id}	GET, PUT, DELETE
Passagier	/person/passagier/	GET, POST
	/person/passagier/{id}	GET, PUT, DELETE
Flugplan	/flugplan/	GET, POST
	/flugplan/{id}	GET, PUT, DELETE
Flug	/flugplan/{id}/flug/	GET, POST
	/flugplan/{id}/flug/{id}	GET, PUT, DELETE
Sitzplatz	/flugplan/{id}/flug/{id}/sitzplatz/	GET, POST
	/flugplan/{id}/flug/{id}/sitzplatz/{id}	GET, PUT, DELETE
Auftragskopf	/auftragskopf/	GET, POST
	/auftragskopf/{id}	GET, PUT, DELETE
Auftragsposition	/a.kopf/{id}/auftragsposition/	GET, POST
	/a.kopf/{id}/auftragsposition/{id}	GET, PUT, DELETE
Lieferbedingung	/lieferbedingung/	GET, POST
	/lieferbedingung/{id}	GET, PUT, DELETE
Lieferung	/lieferung/	GET, POST
	/lieferung/{id}	GET, PUT, DELETE
eTicket	/lieferung/{id}/eTicket/	GET, POST
	/lieferung/{id}/eTicket/{id}	GET, PUT, DELETE
…	…	…

Initiales OS-VD (Ausschnitt)



Initiales Ressourcenschema (VR) (Ausschnitt)

Vorgangsdienst	URL der Ressourcen	HTTP-Operatoren
>Suchanfrage	/>suchanfrage/	GET, POST
	/>suchanfrage/{id}	GET, PUT, DELETE
Flugstatus>	/flugstatus>/	GET, POST
	/flugstatus>/{id}	GET, PUT, DELETE
>Flugstatus	/>flugstatus/	GET, POST
	/>flugstatus/{id}	GET, PUT, DELETE
Flugdaten>	/flugdaten>/	GET, POST
	/flugdaten>/{id}	GET, PUT, DELETE
>Flugdaten	/>flugdaten/	GET, POST
	/>flugdaten/{id}	GET, PUT, DELETE
Suchergebnis>	/suchergebnis>/	GET, POST
	/suchergebnis>/{id}	GET, PUT, DELETE
>Flugwahl	/>flugwahl/	GET, POST
	/>flugwahl/{id}	GET, PUT, DELETE
Angebot>	/angebot>/	GET, POST
	/angebot>/{id}	GET, PUT, DELETE
…	…	…

Initiales Dokumentenschema (Ausschnitt)

Dokumenttyp (p/t)	Dokumentstruktur
Kunde	KundeID: Zahl, Nachname: Zeichenkette, Vorname: Zeichenkette, Kundenadresse: {PLZ: Zeichenkette, Ort: Zeichenkette, Nr: Zeichenkette} …
Flug (Sitzplatz)	FlugID: Zahl, FlugNr: Zeichenkette, Start-/Ankunftszeit: DatumUhrzeit, Start-/ZielflughafenID: Zeichenkette, Sitzplatzliste: {PlatzNr: Zeichenkette, Klasse: Zeichen, ...} …
Leistungsspezifikation	LeistungsspezID: Zahl, Version: Fließzahl, AGB-Text: Zeichenkette, …
Suchanfrage (transient)	Hinflug-/Rückflugdatum: Datum, Hin-/Rückflugzeit: Uhrzeit, Start-/Zielflughafen: Zeichenkette, …
Flugstatus (t)	Dokumenttyp: Suchanfrage (Flugattribute)
Flugdaten (t)	Liste mit verfügbaren Flügen und Sitzplätzen (siehe Dokumenttyp Flug), interne Spezifikationen für Preisberechnung
Suchergebnis (t)	Liste mit verfügbaren Flügen (siehe Dokumenttyp Flug), Preis je Flug: Euro
Flugwahl (Auftrag)	FlugID: Zahl, Passagieranzahl: Zahl, Status: "Flug_gewählt", …
Angebot (Auftrag)	AuftragskopfID: Zahl, FlugID: Zahl, Sitzplatz_frei: [Liste mit PlatzNr], Status: "Angebot_erstellt", …
Auftragsdetails (Auftrag)	AuftragskopfID: Zahl, FlugID: Zahl, Status: "Angebot_bestätigt", Sitzplatz_gewählt: [Liste mit PlatzNr], Liste mit Passagierdaten {Nachname: Zeichenkette, Vorname: Zeichenkette, Passagieradresse: { … }, … }
Auftragszusammen-fassung (Flugbuchung)	AuftragskopfID: Zahl, FlugID: Zahl, Sitzplatz_gewählt: [Liste mit PlatzNr], Passagiere: [Liste mit {Vorname, Nachname}], Preis: Euro, Status: "Auftrag_zusammengefasst", …
…	…

Abbildung 5.14: Die initialen Schemata OS-ED, OS-VD, Ressourcen- und Dokumentenschema der Fallstudie

Die Spezifikation des strukturorientierten *OS-ED* wird in Form eines UML-Klassendiagramms repräsentiert. Ein Wechsel der Repräsentationssprache unterstützt die plattformnahe Spezifikation der REST-Schemata und dient zudem der visuellen Hervorhebung des Übergangs auf die softwaretechnische Ebene. Im Zuge der Transformation werden alle als *persistent* markierten KOTs in EOTs überführt (T2.1, Tabelle 5.4). Jede Beziehung zwischen zwei KOTs wird ent-

sprechend den UML-Beziehungstypen in eine Beziehung zwischen zwei EOTs des Klassendiagramms überführt. Beispielsweise werden die (0,*)-is_part_of-Beziehung zwischen *Auftragskopf* und *Auftragsposition* in eine Aggregation (Kardinalitäten (1,1) und (0,*)), oder die (0,*)-interacts_with-Beziehung zwischen *Flug* und *Auftragskopf* in eine Assoziation (Kardinalitäten (1,1)- und (0,*)) überführt. Das konsolidierte VOS ist der Ausgangspunkt für die Ableitung der initialen Spezifikation des OS-VD. Im Zuge dieses Schrittes werden alle VOTs und IOTs übernommen und in eVOTs sowie R-IOTs transformiert (T2.1, Tabelle 5.5). Bei der Transformation des VOS erfolgen keine weiteren Anpassungen. Das initiale OS-VD zeigt Abbildung 5.14 deshalb nur ausschnittsweise.

Das initiale *Ressourcenschema* beschreibt für jeden transformierten Objekttypen (persistenter KOT und VOT) die öffentliche Dienstschnittstelle in Form einer Individual- und Listenressource, um den Zugriff auf deren elementare CRUD-Operatoren zu realisieren (T2.2, Tabelle 5.6 und Tabelle 5.7). Entsprechend der definierten Transformationsregeln werden strukturelle Beziehungen zwischen Objekttypen in der Ressourcen-URL abgebildet (z. B. is_part_of zwischen *Auftragskopf* und *-position*).

Die ausgetauschten Ressourcen-Repräsentationen der RESTful SOA erfasst das initiale *Dokumentenschema*. Die Transformationsspezifikation sieht grundsätzlich die Bildung eines Dokumenttypen für jeden transienten und persistenten KOT vor. Die Struktur der Dokumenttypen wird aus den Objekttyp-Attributen abgeleitet (T2.2, Tabelle 5.8). Für jeden Ausführungszustand eines KOT muss zudem ein eigener Dokumenttyp definiert werden, um die Nachrichtenstruktur zwischen den VOTs unterschiedlicher VOS beschreiben zu können. Die Attributdefinitionen der normalisierten KOTs dienen in der Fallstudie als Basis für die Ableitung der Dokumenttypen von zusammengefassten Objekttypen (vgl. K1.1, [WoBe13, S. 7]). Beispielsweise ist für jeden Ausführungszustand von *Auftragskopf* und *-position* eine Zustandsrepräsentation erforderlich, die durch die Interaktion zwischen Kunde und Vertrieb ausgetauscht werden (z. B. Nachricht *Flugwahl (a)*, *Angebot (a)*, etc.).

5.5.5 Konsolidierung des REST-Architekturmodells (K2)

Auf Basis der initial abgeleiteten Schemata des REST-Architekturmodells wird die softwaretechnische Spezifikation der RESTful SOA im Entwicklungsschritt *Konsolidierung K2* erarbeitet. Die Konsolidierung erfolgt in vier Teilschritten (K2.1-K2.4), die sukzessive durchlaufen werden und mit der Überarbeitung der Spezifikationen von OS-ED, OS-VD und Dokumentenschema sowie der Definition des Datenschemas einhergehen. Das Ressourcenschema wird parallel zur Konsolidierung von OS-ED und OS-VD überarbeitet. Als Ergebnis einer erfolgreichen Durchführung des Entwicklungsschrittes K2 liegen die konsolidierten Schemata OS-ED, OS-VD, Ressourcenschema, Dokumentenschema sowie das Datenschema vor (vgl. Metamodelle der Abschnitte 5.3.3 bis 5.3.6).

Zielsetzung	Konsolidierung der softwaretechnischen Spezifikation der RESTful SOA (K2)
Input	Initiales OS-ED, initiales OS-VD, initiales Ressourcenschema, initiales Dokumentenschema
Output	K2.1: Konsolidiertes OS-ED, konsolidiertes Ressourcenschema (ER) K2.2: Konsolidiertes OS-VD, konsolidiertes Ressourcenschema (VR) K2.3: Konsolidiertes Dokumentenschema K2.4: Strukturdefinition Datenschema
Lösungsansatz	Spezifikation von Schritten für die Durchführung einer methodisch geleiteten Überarbeitung der Schemata (nicht-automatisiert).

Tabelle 5.9: Zusammenfassung von Entwicklungsschritt *Konsolidierung K2*

Das Ergebnis der zweiten Konsolidierung ist die softwaretechnische Spezifikation der RESTful SOA aus Innen- und Außensicht. Sie stellt die Quelle für die Ableitung des initialen Implementierungsmodells dar (Transformation T3, Abschnitt 5.5.6). Die Durchführung von K2 entspricht den Aktivitäten *detaillierter Serviceentwurf* und *Entwurf Gesamtsystem* des Entwicklungsvorgehens der SOM-R-Methodik (Abschnitt 5.4). Hierbei ist ein Aufdecken weiterer Dienste möglich. Eine weiterführende Diskussion der Konzeptionsentscheidungen von Entwicklungsschritt K2 findet sich in Anhang B.2.3.

5.5.5.1 Konsolidierung des OS-ED und Ressourcenschemas (ER) (K2.1)

Im ersten Teilschritt K2.1 erfolgt die Überarbeitung des initialen OS-ED und des initialen Ressourcenschemas der Entitätsressourcen. Gegenstand der Konsolidierung sind somit die strukturorientierten Schemata der Anwendungsfunktionen des REST-Architekturmodells. Das Vorgehen sieht eine strukturelle Überarbeitung der Systemspezifikation (Innensicht) und anschließend ihre Abstimmung mit der öffentlichen REST-Schnittstelle (Außensicht) vor.

Die **Konsolidierung der Struktur** (Bausteine und Beziehungen) **des initialen OS-ED und Ressourcenschemas (ER)** (abgekürzt *RS-ER*) umfasst folgende Schritte:

1. Hinzufügen von Beziehungen zur expliziten Erfassung von transitiven Verknüpfungen zwischen existenzunabhängigen und/oder existenzabhängigen EOTs (OS-ED).
2. Definieren von Einstiegspunkten in die RESTful SOA (OS-ED, RS-ER).
3. Entfernen von Entitätsressourcen des initialen Ressourcenschemas, die nicht die Funktionalität eines RESTful Services veröffentlichen (RS-ER).
4. Aufdecken und spezifizieren von Entitätsressourcen zur Abbildung der Funktionalität fachlicher Methoden von EOTs auf die REST-Schnittstelle (RS-ER).
5. Spezifizieren zusätzlicher Ressourcen-Repräsentationen, die Parameter öffentlicher Methoden sind und als Nachrichten von den Ressourcen empfangen (Eingabe, IN) oder gesendet (Rückgabe, OUT) werden (OS-ED, RS-ER).
6. Definieren von internen sowie externen Verlinkungen (Links) und Aktualisieren der Ressourcen-Repräsentationen mit diesen Links.
   Hinweis: Die definierten Ressourcen-Repräsentationen (Ergebnis von Schritt 6) gehen als Input in die *Konsolidierung des Dokumentenschemas* ein (K2.3, Abschnitt 5.5.5.3).

Die **Konsolidierung der Eigenschaften der Modellelemente des initialen OS-ED und Ressourcenschemas (ER)** umfasst folgende Überarbeitungsschritte:

7. Typisieren der Attribute (OS-ED).
8. Identifizieren und spezifizieren anwendungsspezifischer Datentypen (OS-ED).
9. Typisieren der Methodenparameter (Eingabe/Rückgabe) (OS-ED).
10. Entfernen von (HTTP-)Methoden der initial abgeleiteten Ressourcen, die für eine Realisierung der Funktionalität von Entitätsdiensten nicht benötigt werden (OS-ED, RS-ER).
11. Definieren der Ausnahmebehandlung in Methoden (OS-ED) sowie bestimmen von Response Codes zur Beschreibung des Verarbeitungsstatus einer Anfrage in Antwortnachrichten öffentlicher Methoden (OS-ED, RS-ER).
12. Zuordnen akzeptierter Medientypen zu öffentlichen Methoden (OS-ED, RS-ER).
13. Konsistente Abstimmung der Spezifikation der REST-Schnittstelle mit den Überarbeitungsergebnissen nach den Schritten 10-12 (RS-ER).

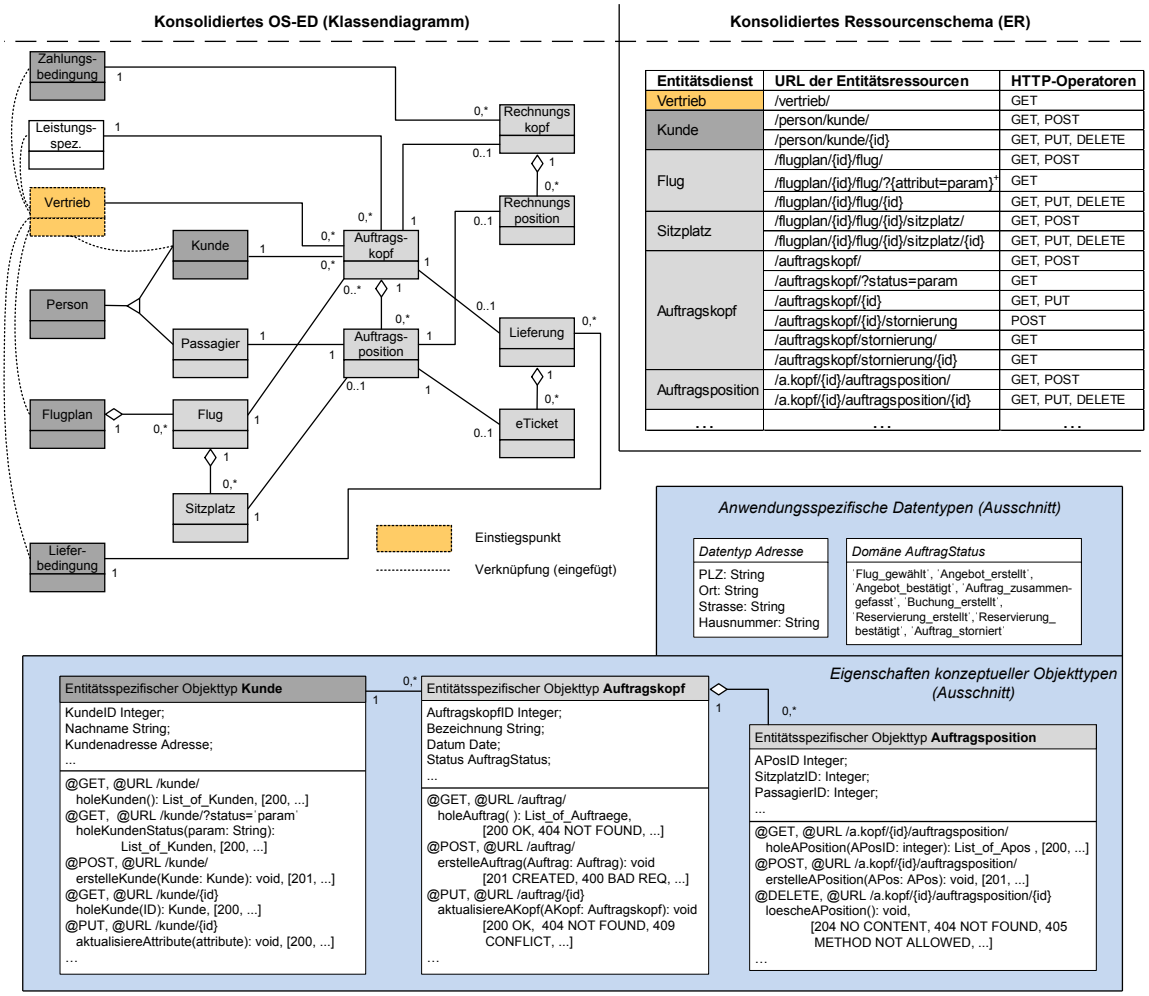
Ergebnis des ersten Teilschrittes K2.1 sind das *konsolidierte OS-ED* sowie das *konsolidierte Ressourcenschema (ER)*. Die Anwendung des Entwicklungsschrittes K2.1 zeigt das nachfolgende Beispiel der Fallstudie.

Weitere Anforderungen an die Überarbeitung der Spezifikationen von Objekttypen des OS-ED und ER von RS-ER können sich z. B. aus Ergebnissen einer Konsolidierung des OS-VD ergeben und ein mehrmaliges Durchlaufen der Überarbeitungsschritte erfordern. Die Schritte 1 und 2 wurden erstmalig in [Wolf12, S. 1656 f.] vorgeschlagen. Eine Erläuterung der bei der Konzeption von Entwicklungsschritt K2.1 getroffenen Entscheidungen findet sich in Anhang B.2.3.

BEISPIEL FLUGBUCHUNG (KONSOLIDIERTES OS-ED UND RESSOURCENSCHEMA (ER))

Im ersten Schritt werden durch das Hinzufügen von Beziehungen die (relevanten) transitiven Abhängigkeiten zwischen dem Vertrieb und den weiteren linksstehenden EOTs explizit modelliert. Anschließend wird der EOT *Vertrieb* als Einstiegspunkt bestimmt, von dem aus weitere Entitätsdienste über Verknüpfungen zugreifbar sind (K2.1, Schritte 1-2). Im Ressourcenschema wird die Individualressource *Vertrieb* entfernt (Schritt 3). Der Entitätsdienst *Auftragskopf* realisiert den Zugriff auf seine unterschiedlichen Zustände durch eine neue Filterressource. Eine weitere Filterressource ermöglicht die Selektion von Flügen durch Angabe von Suchparametern für ein bis beliebig viele Elemente der Attribute des Objekttyps *Flug* (*flug/?{attribut=param}*$^{+}$). Zudem soll die fachliche Funktionalität der *Stornierung eines Auftrags* über eine eigene Ressource zugreifbar sein. Die Liste stornierter Aufträge erfasst eine eigene Ressource (*/auftragskopf/stornierung*). Sie wird zudem als Ausprägung in den Auftragsstatus aufgenommen (Schritt 4). Die Spezifikation zusätzlicher Repräsentationen von Ressourcen, die als Teil der Aufruf- oder Antwortnachrichten von HTTP-Methoden empfangen und gesendet werden, erfolgt nicht. Sie entsprechen im Wesentlichen den Dokumenttypen des initialen Dokumentenschemas und werden in Abschnitt 5.5.5.4 final dargestellt. Die Verknüpfungen, die Dienste durch ihre Repräsentationen bereitstellen, orientieren sich an den definier-

ten Beziehungen zwischen den Objekttypen des OS-ED (Schritte 5 und 6). Das konsolidierte OS-ED und das konsolidierte Ressourcenschema (ER) sind in Abbildung 5.15 dargestellt.



Entitätsdienst	URL der Entitätsressourcen	HTTP-Operatoren
Vertrieb	/vertrieb/	GET
Kunde	/person/kunde/	GET, POST
	/person/kunde/{id}	GET, PUT, DELETE
Flug	/flugplan/{id}/flug/	GET, POST
	/flugplan/{id}/flug/?{attribut=param}+	GET
	/flugplan/{id}/flug/{id}	GET, PUT, DELETE
Sitzplatz	/flugplan/{id}/flug/{id}/sitzplatz/	GET, POST
	/flugplan/{id}/flug/{id}/sitzplatz/{id}	GET, PUT, DELETE
Auftragskopf	/auftragskopf/	GET, POST
	/auftragskopf/?status=param	GET
	/auftragskopf/{id}	GET, PUT
	/auftragskopf/{id}/stornierung	POST
	/auftragskopf/stornierung/	GET
	/auftragskopf/stornierung/{id}	GET
Auftragsposition	/a.kopf/{id}/auftragsposition/	GET, POST
	/a.kopf/{id}/auftragsposition/{id}	GET, PUT, DELETE
...	...	...

Abbildung 5.15: Konsolidiertes OS-ED und Ressourcenschema (ER) der Fallstudie

Das Ergebnis einer Spezifikation der Bausteineigenschaften wird in der unteren Hälfte von Abbildung 5.15 für die EOTs *Kunde*, *Auftragskopf* und *Auftragsposition* dargestellt. So werden die Attribute typisiert, anwendungsspezifische Datentypen definiert und die Ein-/Ausgabeparameter von Methoden typisiert (Schritte 7-9). Die Spezifikation der öffentlichen REST-Schnittstelle erfolgt parallel zu den genannten Überarbeitungen. Diese sieht z. B. das Entfernen nicht benötigter HTTP-Methoden, die Definition des Ausnahmeverhaltens von Methoden und die Zuordnung von Response Codes sowie akzeptierter Medientypen zu Objekttypen und ihren Ressourcen vor (Schritte 10-12). Für eingehende Nachrichten werden standardmäßig Zustandsinformationen in den Formaten JSON und XML akzeptiert. Response-Nachrichten können ihre Antworten z. B. als JSON- und HTML-Dokument bereitstellen. Abschließend wird eine Abstimmung mit der definierten Ressourcenschnittstelle durchgeführt.

5.5.5.2 Konsolidierung des OS-VD und Ressourcenschemas (VR) (K2.2)

Die strukturelle Überarbeitung und Anreicherung von initialem OS-VD und initialem Ressourcenschema (VR) des REST-Architekturmodells ist Gegenstand von Konsolidierungsschritt K2.2. Beide Schemata beschreiben die verhaltensorientierte Sicht auf die Anwendungsfunktionen

sowie den Kommunikationsteil des AwS. Mit der Schema-Konsolidierung wird zudem die Spezifikation der Vorgangsdienste der RESTful SOA sowie deren Koordinationsbeziehungen angestrebt. Jeder Vorgangsdienst (VD) umfasst eine Menge von fachlich zusammengehörigen Vorgangsressourcen (VR), die durch einen oder mehrere VOTs eines OS-VD realisiert werden (Abschnitt 3.3.3.2).

Das Entwicklungsvorgehen sieht zunächst eine Überarbeitung der Struktur von OS-VD sowie der VOTs (Innensicht) und parallel dazu die Abstimmung der öffentlichen Schnittstelle (Außensicht) der bereitgestellten Vorgangsressourcen vor (Teilschritt K2.2.1). Anschließend wird die Spezifikation der R-IOTs verfeinert (K2.2.2).

GESTALTUNG DER SERVICE-KOORDINATION

Im Kontext einer Konsolidierung der verhaltensspezifischen Schemata des REST-Architekturmodells steht der Entwurf der Koordination von Vorgangsdiensten besonders im Fokus. Die Bestimmung von Gestaltungsalternativen zur Realisierung der Service-Koordination ist dabei wichtige Voraussetzung, um Konsolidierung K2.2 konzipieren zu können.

Grundsätzlich werden in SOA die Formen der nicht-hierarchischen Koordination zwischen autonomen Diensten (Choreographie) und der hierarchischen Koordination von Diensten durch eine zentrale Steuerungsinstanz (Orchestrierung) unterschieden (Abschnitt 3.1.4). Da im initialen OS-VD die Koordinationsbeziehungen zwischen abgeleiteten eVOTs standardmäßig als nicht-hierarchisch spezifiziert sind (T2.1), muss als Voraussetzung für die weitere Entwicklung eine Entscheidung für oder gegen den Entwurf einer hierarchischen Koordination getroffen werden.

Die eVOTs eines initialen OS-VD bilden den Ausgangspunkt, um die Koordinationsbeziehungen zwischen RESTful Services in der SOM-R-Methodik zu bestimmen. Im Falle der Einführung einer hierarchischen Koordinationsstruktur sind verschiedene Ansätze zur Aufdeckung nicht-elementarer Vorgangsdienste denkbar. Für die *Gestaltung der hierarchischen Service-Koordination* werden im Folgenden drei **grundlegende Alternativen** vorgestellt:

- *Spezifizieren eines nicht-elementaren Vorgangsdienstes für jedes OS-VD*: Die Orchestrierung der elementaren Vorgangsdienste eines OS-VD erfolgt hier durch einen nicht-elementaren Vorgangsdienst. Übertragen auf die Modellebene des SOM-GPM wird jedes *betriebliche Objekt* in einen nicht-elementaren VD überführt. Die zu erstellenden Vorgangsressourcen des neuen Dienstes könnten in diesem Fall durch einen neuen oder auch einen existierenden Objekttypen des REST-Architekturmodells realisiert werden.
- *Identifizieren nicht-elementarer Vorgangsdienste durch das Abgrenzen zusammenhängender Vorgangsnetze*: Ein nicht-elementarer VD steuert die Ausführung einer Menge von elementaren VDs eines OS-VD, die mit der Durchführung von Vorgängen in Vorgangsnetzen eines betrieblichen Objektes korrespondieren (vgl. Abschnitte 2.1.3 und 3.1.4). Das Aufdecken zusammenhängender Vorgangsnetze, und damit auch die Identifikation von nicht-elementaren VDs, kann durch die Analyse der *Transaktionsbeziehungen* unterstützt werden. Eine zusätzliche Entscheidungshilfe stellt hierbei die

Anwendung von *Message Exchange Patterns* (MEP) [CHL+07], wie dem MEP *In-Out*, dar [KrSi11, S. 299].

- *Vollständig nicht-hierarchische Koordination von Vorgangsdiensten*. Die Choreographie des Gesamtsystems basiert auf dem Verständnis des Ablaufs einer Prozesskette, in der jeder Vorgangsdienst, ähnlich der Aktivität innerhalb eines Prozesses, nur seinen oder seine möglichen Nachfolger kennt. Die Prozessausführung wird in Form von Ein-Weg-Aufrufen koordiniert [Josu08, S. 122 f.]. Dieser Ansatz entspricht dem OS-VD, das in seiner initial abgeleiteten Spezifikation den nachrichtengesteuerten Ablauf von eVOTs definiert.

Neben den genannten Lösungsansätzen bestehen weitere Alternativen zum Entwurf der Koordinationsbeziehungen zwischen den Diensten einer SOA. Grundsätzlich sind auch Formen, wie z. B. die Orchestrierung von elementaren VDs mehrerer oder sogar aller OS-VD des zu entwickelnden AwS durch einen *globalen, zentralen Koordinator* denkbar**.** Zudem können weitere Mischformen oder Spezialisierungen der vorgestellten Gestaltungsvarianten gebildet werden.

Im Folgenden wird der Entwicklungsschritt zur Konsolidierung von Vorgangsdiensten mit der SOM-R-Methodik vorgestellt. Bezüglich des Entwurfs der Service-Koordination werden Freiheitsgrade definiert, um die nicht-hierarchischen und/oder hierarchischen Koordinationsbeziehungen von VDs gemäß den Anforderungen des Entwicklungsprojektes realisieren zu können.

KONSOLIDIERUNG DES OS-VD UND RESSOURCENSCHEMAS (VR) (K2.2.1)

In Teilschritt K2.2.1 erfolgt die Überarbeitung des initialen OS-VD und initialen Ressourcenschemas (VR). Im Fokus der Entwicklung steht die Überarbeitung der Struktur und Eigenschaften von VOTs und VRs der elementaren Vorgangsdienste der RESTful SOA. Des Weiteren ist zu entscheiden, ob nicht-elementare Vorgangsdienste aufgedeckt und spezifiziert werden sollen.

Die Überarbeitungsschritte von K2.2.1 zur **Konsolidierung des initialen OS-VD und initialen Ressourcenschemas (VR)** (abgekürzt: *RS-VD*) lauten:

1. Identifizieren und spezifizieren elementarer Vorgangsdienste:
   1.1 Abgrenzen elementarer Vorgangsdienste, die die Realisierung einer fachlich abgeschlossenen Funktionalität der Geschäftslogik kapseln. Die Quelle für die Abgrenzung bilden die elementaren VOTs und Interaktionsbeziehungen des OS-VD. Als Kriterium für die Identifikation und Abgrenzung von VDs werden MEP mit den Mustern *In-Out* (*Request-Response*) und *In-Only* herangezogen, über welche die Kommunikation zwischen Servicenutzer und Serviceanbieter beschrieben wird (in Anlehnung an [KrSi11, S. 299], [WoBe13, S. 8]).
   1.2 Entscheidung über das Zusammenfassen von elementaren VOTs im OS-VD. Neu abgegrenzte elementare VDs (Schritt 1.1) werden aus der Innensicht durch einen oder mehrere eVOTs realisiert. Korrespondierend zur Neugestaltung der Vorgangsdienste können diese eVOTs ebenfalls zusammengefasst werden.
   1.3 Spezifizieren von Attributen und Methoden der neu gebildeten eVOTs (Innensicht). Die Basis hierfür bilden die initial abgeleiteten Eigenschaften der Objekttypen.

1.4 Initiales Ableiten der Vorgangsressourcen und ihrer REST-Schnittstelle (Identifikator, HTTP-Operatoren) für die neu identifizierten elementaren VDs (Außensicht). Die Basis hierfür bildet das initiale Ressourcenschema (VR).

2. Wahl und Entwurf der *Koordination von Vorgangsdiensten* (vgl. Unterabschnitt „Gestaltung der Service-Koordination“). Zwei klassische Alternativen, die auch in einer Mischform gebildet werden können, sind:
   - Gestaltung der *Service-Orchestrierung*. Die Schritte zum Aufdecken von nicht-elementaren Vorgangsdiensten und dem Entwurf der hierarchischen Koordinationsbeziehungen zwischen den Diensten sind:
     2.1 Identifizieren von nicht-elementaren VDs, die das Lösungsverfahren zur Orchestrierung einer Menge von elementaren VDs in einem neuen neVOT kapseln und über die REST-Schnittstelle ihrer Ressourcen bereitstellen (Ausgangspunkt OS-VD).
        - Jeder nicht-elementare Vorgangsdienst wird aus der Innensicht durch einen neVOT realisiert und ist über interacts_with-Beziehungen mit den gesteuerten eVOTs verknüpft.
        - Jeder nicht-elementare Vorgangsdienst veröffentlicht eine Menge von Vorgangsressourcen.

     2.2 Spezifizieren von Attributen und Methoden der neu identifizierten neVOTs (Innensicht nicht-elementarer VDs).

     2.3 Ableiten der REST-Schnittstelle (Identifikator, HTTP-Operatoren) als Menge von Vorgangsressourcen nach den Regeln in Tabelle 5.10 (Außensicht).

OS-VD	Ressourcentyp (neVR)	Identifikator (URL)	Operator (HTTP)
Nicht-elementarer Vorgangsobjekttyp	Listenressource	/ressource/	GET, POST
	Individualressource	/ressource/{id}	GET, PUT, DELETE

Tabelle 5.10: Ableitungsregeln für die nicht-elementaren Vorgangsressourcen

   - Gestaltung der *Service-Choreographie*. Jedes initial abgeleitete OS-VD beschreibt für jeweils ein betriebliches Objekt die automatisierte Durchführung von Vorgängen eines konkreten Geschäftsvorfalls als Menge lose gekoppelter eVOTs. Diese sind untereinander oder mit der Umwelt durch interacts_with-Beziehungen verknüpft. Der hierbei spezifizierte nachrichtengesteuerte Ablauf der eVOTs geht aus dem ereignisgesteuerten Ablauf von Vorgängen im VES hervor. Eine gesonderte Überarbeitung der Modellstruktur des OS-VD ist damit nicht notwendig.

3. Bestimmen von Einstiegspunkten in die RESTful SOA über Vorgangsressourcen. Die Interaktionsbeziehungen zwischen einem OS-VD und seiner Umwelt geben Hinweise auf geeignete Einstiegspunkte in Abstimmung mit K2.1, Schritt 2 (OS-VD, RS-VR).

4. Hinzufügen von Beziehungen zwischen VOTs zur Beschreibung logischer Verknüpfungen, die Ablaufbeziehungen im Rahmen der nicht-hierarchischen oder hierarchischen Koordination realisieren und zudem als Hyperlinks in den Entwurf der Hypermedia einfließen (OS-VD).

5. Entfernen von Vorgangsressourcen, die durch den RESTful Service nicht veröffentlicht werden (RS-VR). Dieser Schritt betrifft primär die initial abgeleiteten Vorgangsressourcen.
6. Aufdecken und spezifizieren von Vorgangsressourcen zur Abbildung der Funktionalität von fachlichen Methoden von VOTs auf die REST-Schnittstelle (RS-VR).
7. Spezifizieren weiterer Ressourcen-Repräsentationen, die Parameter öffentlicher Methoden repräsentieren und in Form von Nachrichten durch diese empfangen (Eingabe, IN) oder gesendet (Rückgabe, OUT) werden (OS-VD, RS-VR).
8. Definieren von internen sowie externen Links und aktualisieren der Repräsentationen von Vorgangsressourcen (OS-VD, RS-VR).
   Hinweis: Als Ergebnis von Schritt 8 gehen die spezifizierten Ressourcen-Repräsentationen als Input in die *Konsolidierung des Dokumentenschemas* ein (Abschnitt 5.5.5.3).

Aufbauend auf der überarbeiteten Struktur des OS-VD und der neuen Vorgangsressourcen werden die Eigenschaften der identifizierten Vorgangsdienste spezifiziert.
Die Überarbeitungsschritte von K2.2.1 zur **Konsolidierung von Eigenschaften der Bausteine des OS-VD und RS-VR** sind:

9. Typisieren der Attribute (OS-VD).
10. Typisieren der Methodenparameter (OS-VD, RS-VR).
11. Entfernen der Methoden von Vorgangsressourcen, die keine öffentlichen Funktionen realisieren (OS-VD, RS-VR).
12. Spezifizieren der Lösungsverfahren von VOTs im REST-Architekturmodell. Die überführten Lösungsverfahren der VOTs der fachlichen AwS-Ebene (konsolidiertes VOS) sind die Basis zur Spezifikation der Ausführungslogik der Vorgangsdienste in Form von Workflows (OS-VD).
13. Gestalten der eingehenden und ausgehenden Nachrichten von Vorgangsressourcen durch Zuordnung von Parametern und Dokumenten (in Abstimmung mit der Konsolidierung des Dokumentenschemas (K2.3, S. 141)). Die Definition der Dokumentstruktur erfolgt in Abstimmung mit den Repräsentationen der Entitäts- und Vorgangsressourcen (vgl. K2.2.1, Schritte 7 und 8; K2.1, Schritte 5 und 6).
14. Definieren der Ausnahmebehandlung in Methoden (OS-VD) sowie Bestimmung von Response Codes zur Beschreibung des Verarbeitungsstatus einer Anfrage in den Antwortnachrichten der öffentlichen Methoden (OS-VD, RS-VR).
15. Zuordnen von Medientypen zu öffentlichen Methoden (OS-VD, RS-VR).
16. Abstimmen der Ressourcenschnittstelle mit den Schritten 12-15 (RS-VR).
17. Definieren fachlicher Transaktionen zur Kompensation von Workflows (Transaktionssicherheit).

Die Konzeption des Entwicklungsschrittes K2.2.1 wird in Anhang B.2.3 genauer erläutert und einzelne Entwurfsentscheidungen näher begründet. Das Ergebnis der Konsolidierung K2.2.1

sind das konsolidierte Ressourcenschema (VR) sowie die konsolidierten VOTs des OS-VD. Nachfolgend werden die Schritte zur Überarbeitung der Interface-Objekttypen des OS-VD vorgestellt.

KONSOLIDIERUNG DES INTERFACEOBJEKTSCHEMAS (K2.2.2)

Die Interface-Objekttypen der REST-Ebene (R-IOT) beschreiben die Kommunikationskanäle der RESTful SOA zu andersartigen Aufgabenträgern (der Diskurs oder Umwelt). R-IOTs kapseln hierfür die Realisierung der Nutzerkommunikation und spezifizieren u. a. den Aufbau der Präsentationsschicht, das eingesetzte Kommunikationsprotokoll sowie die Attribute und Methoden zur Verarbeitung der empfangenen und bereitgestellten Informationen oder auch die Formatierung des Inhalts der ausgetauschten Nachrichten. Zu diesem Zweck erfolgt auf der fachlichen Ebene ein Zuordnen von IOTs zu VOTs und die initiale Beschreibung ihrer Eigenschaften (Abschnitt 5.5.3). Das IOS wird nachfolgend an das konsolidierte OS-VD angepasst und um softwaretechnische Merkmale angereichert.

Die Schritte zur **Konsolidierung der Schnittstellenspezifikation des OS-VD** lauten:

1. Entfernen von R-IOTs, die, als Ergebnis der Konsolidierung des OS-VD, für eine externe Kommunikation nicht mehr benötigt werden.
2. Hinzufügen oder zusammenfassen von R-IOTs, um Kommunikationskanäle für neu identifizierte bzw. zusammengefasste VOTs bereitzustellen (vgl. K2.2.1, Schritt 2).
3. Spezifizieren der Schnittstellenmerkmale und Eigenschaften der neuen R-IOTs.
4. Bestimmen des Kommunikationsprotokolls und der Anforderungen an die Formatierung der Nachrichten, die ein VOT jeweils sendet oder empfängt (in Abstimmung mit K2.1 (OS-ED) und K2.2.1 (OS-VD)).
5. Zusammenfassen von R-IOTs, die von mehreren VOTs gemeinsam genutzt werden können. Dieser Schritt dient der Vermeidung von Funktionsredundanz.
6. Typisieren von Attributen.
7. Typisieren von Methodenparametern.
8. Definieren von Ausnahmenverhalten.

Das Ergebnis des zweiten Teilschrittes *Konsolidierung K2.2* sind das konsolidierte OS-VD und das konsolidierte Ressourcenschema (VR). Die Überarbeitung der Schemaspezifikationen erfolgt in Abstimmung mit der Konsolidierung des OS-ED. Die Überarbeitungsschritte können dabei mehrmals durchlaufen werden.

BEISPIEL FLUGBUCHUNG (KONSOLIDIERTES OS-VD UND RESSOURCENSCHEMA (VR))

Die Struktur des initialen OS-VD der Fallstudie wird im Zuge der Konsolidierung überarbeitet und elementare Vorgangsdienste anhand der initialen eVOTs identifiziert (vgl. Abbildung 5.14). Das Ergebnis der Zusammenfassung von eVOTs (K2.2.1, Schritt 1) zeigt Tabelle 5.11 für den betrachteten Ausschnitt der Fallstudie.

OS-VD	eVOT (initiales OS-VD)	eVOT (überarbeitet)	*Überarbeitungsschritt (K2.2.1)*
Vertrieb	>Suchanfrage	Flugsuche	Zusammenfassen (MEP: In-Out)
	Flugstatus>		
	>Flugdaten		
	>Suchergebnis		
	>Flugwahl	Angebots-abwicklung	Zusammenfassen (MEP: In-Out)
	Angebot>		
	>Auftragsdetails	Auftragserstellung	Zusammenfassen (MEP: In-Out)
	Auftragszusammenf.>		
	...	...	...
Flugbetrieb	>Flugstatus	Flugauskunft	Zusammenfassen (MEP: In-Out)
	Flugdaten>		
...	...	...	...

Tabelle 5.11: Konsolidierung des OS-VD der Fallstudie (Ausschnitt)

Als Kriterium für die Abgrenzung von Diensten im OS-VD wird im Ausschnitt der Fallstudie das MEP *In-Out* herangezogen (Schritt 1). Die Koordination der Vorgangsdienste erfolgt in nicht-hierarchischer Form (Schritt 2). Als elementare Vorgangsdienste werden *Flugsuche*, *Angebots-abwicklung* und *Auftragserstellung* des OS-VD *Vertrieb* sowie die *Flugauskunft* des OS-VD *Versand* identifiziert. Die abgegrenzten Vorgangsdienste stellt das konsolidierte OS-VD der Fallstudie in Abbildung 5.16 dar.

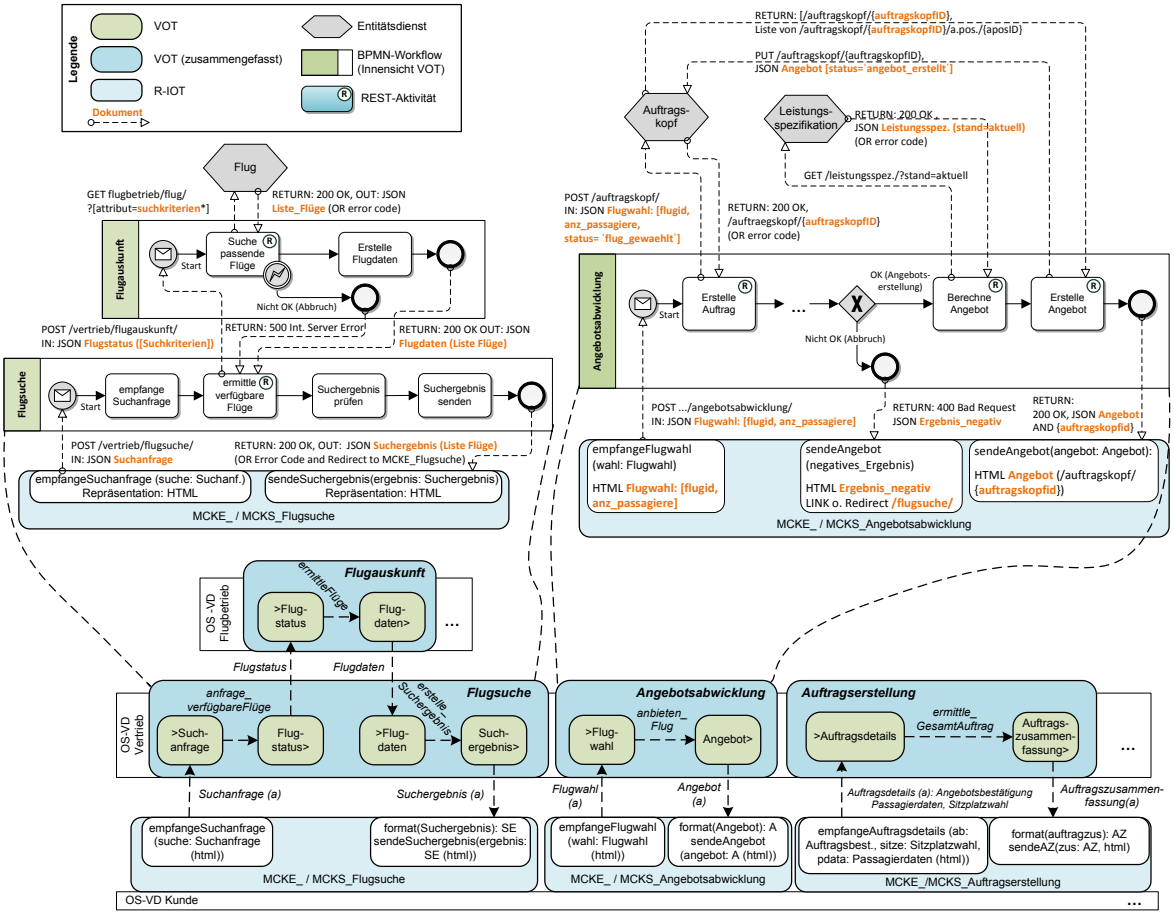


Abbildung 5.16: Konsolidiertes OS-VD der Fallstudie (Ausschnitt)

Als Einstiegspunkt in das Flugbuchungssystem wird, in Abstimmung mit dem OS-ED, die Ressource *Vertrieb* im OS-VD gewählt (Schritt 3). Durch den Aufruf URL */vertrieb/* werden den externen Nutzern die Links zum Aufruf verfügbarer Funktionen der RESTful Vorgangsservices in einem Dokument, z. B. die Kundendaten als HTML-Dokument, bereitgestellt. Im Rahmen der Servicenutzung sind Rücksprünge auf die VOTs eines früheren Schritts im Geschäftsablauf grundsätzlich vorgesehen, um z. B. von der Angebotsabwicklung auf eine neue Flugsuche zu gelangen (Schritt 4).

Korrespondierend zu den Ergebnissen der ersten Überarbeitungsschritte werden nicht benötigte VRs entfernt und neue VRs abgeleitet (Schritte 5 und 6). Beispielsweise werden die Ressourcen zur Nutzung des neu gebildeten Dienstes *Flugsuche* bereitgestellt. Die initial abgeleiteten Ressourcen von */>Suchanfrage/*, */Flugstatus>/*, */>Flugdaten/* und */>Suchergebnis/* werden dagegen entfernt. Auf Basis der konsolidierten Struktur des OS-VD werden die Ressourcen-Repräsentationen, die von öffentlichen Methoden gesendet und empfangen werden, überarbeitet und den Aufrufbeziehungen der Workflowspezifikationen zugeordnet (Schritte 7 und 8). Zum Beispiel wird eine neue Suche über POST auf */vertrieb/flugsuche/* gestartet. Die Suchparameter werden in dem JSON-Dokument *Suchanfrage* übergeben.

Die Konsolidierung der Eigenschaften beginnt mit der Typisierung von Attributen und den Ein-/Ausgabeparametern von Methoden (Schritte 9 und 10). Als HTTP-Methoden wird lediglich das POST der Ressource *Vertrieb* entfernt (Schritt 11). Auf Basis der übernommenen Lösungsverfahren der fachlichen Ebene erfolgt die Erstellung der Lösungsverfahren von VOTs (Schritt 12). Die Lösungsverfahren werden in der vorgeschlagenen RESTful Erweiterung der Workflowsprache BPMN spezifiziert (Abschnitt 5.3.6). Jede *REST-Aktivität* kapselt die Logik zur Durchführung eines REST-Aufrufs (z. B. „ermittle verfügbare Flüge“). Die Parameter eines *REST-Aufrufs* werden als textuelle Annotation des Nachrichtenflusses modelliert. Beispielsweise wird eine Workflowinstanz von Dienst *Angebotsabwicklung* mit POST auf */angebotsabwicklung/* und der Übergabe des Dokuments *Flugwahl* gestartet. Das spezifizierte Lösungsverfahren sieht zunächst das Erstellen eines Auftrags auf Basis der empfangenen Daten vor. Ein neuer Auftragskopf mit Status „Flug_gewaehlt“ wird durch POST auf */auftragskopf/* und Übergabe von Flugwahl realisiert. Nach positiver Rückmeldung (200 OK) wird für die Anzahl der Passagiere jeweils eine Auftragsposition angelegt, die aktuelle Leistungsspezifikation abgefragt und das Angebot berechnet. Im Falle eines negativen Ausführungsergebnisses wird die Kundenabfrage mit einer Fehlermeldung beantwortet (400 BAD REQUEST) und ein Link auf den Dienst *Suchanfrage* zurückgeliefert bzw. ein Redirect durchgeführt.

Parallel zum Lösungsverfahren werden die zwischen den Ressourcen ausgetauschten Nachrichten sowie die Ausnahmebehandlung im Workflow spezifiziert (Schritte 13 und 14). So wird z. B. im Lösungsverfahren der Flugauskunft ein fehlerhafter Aufruf des Entitätsdienstes */Flug/* abgefangen und als Response Code des Aufrufs (505 INTERNAL SERVER ERROR) an die Flugsuche zurückgemeldet. Dort erfolgt eine weitere Behandlung des Fehlers.
HTML und JSON werden als Medientypen der Methodenparameter der REST-Schnittstelle bestimmt und in den R-IOTs dargestellt (Schritt 15). Abschließend wird eine konsistente Abstimmung des Ressourcenschemas mit dem konsolidierten OS-VD vorgenommen (Schritt 16). Auf die Spezifikation von fachlichen Transaktionen (Schritt 17) mit Hilfe von Kompen-

sationsereignissen und -aktivitäten wird an dieser Stelle verzichtet. Es wird angenommen, dass ein erstellter Auftrag nicht mehr gelöscht werden soll (im Falle eines Abbruchs erfolgt eine Stornierung).

Das konsolidierte Ressourcenschema (VR) der Fallstudie zeigt Tabelle 5.12. Über die Einstiegsseite */vertrieb/* werden dem Nutzer des Dienstes *Flugsuche* oder seine erteilten *Aufträge* gemäß ihrem Ausführungszustand bereitgestellt.

Vorgangsdienst	URL der Vorgangsressourcen	HTTP-Operatoren	Request (IN), Response (OUT)
Vertrieb	/vertrieb/	GET	OUT: Einstiegsseite (HTML, JSON)
Flugsuche	/flugsuche/	*GET*	OUT: Definition VD-Flugsuche
		POST	IN: Flugsuche, OUT: Suchergebnis
	/flugsuche/{id}	GET*, PUT, DELETE*	Verwaltung Workflowinstanz
Flugauskunft	/flugauskunft/	*GET*	OUT: Definition VD-Flugauskunft
		POST	IN: Flugstatus, OUT: Flugdaten
	/flugauskunft/{id}	GET*, PUT, DELETE*	Verwaltung Workflowinstanz
Angebots-abwicklung	/angebotsabwicklung/	*GET*	OUT: Definition VD-Angebotsabw.
		POST	IN: Flugwahl, OUT: Angebot
	/angebotsabwicklung/{id}	GET*, PUT, DELETE*	Verwaltung Workflowinstanz
Auftrags-erstellung	/auftragserstellung	*GET*	OUT: Definition VD-Auftragserst.
		POST	IN: Auftragsdetails, OUT: Auftragszusammenfassung
	/auftragserstellung/{id}	GET*, PUT, DELETE*	Verwaltung Workflowinstanz
…	…	…	…

Tabelle 5.12: Konsolidiertes Ressourcenschema (VR) der Fallstudie (Ausschnitt)

Ein personeller Nutzer startet den Geschäftsablauf durch Eingabe von *Suchkriterien*, die mittels POST an den Dienst */flugsuche/* gesendet werden. Die Daten einer (beendeten) Workflowinstanz sind über GET aufrufbar. Weitere Methoden der REST-Schnittstelle dienen dem Abfragen der Workflowdefinition/-informationen oder der Verwaltung von Workflowinstanzen. Sie stehen externen Nutzern deshalb nicht zur Verfügung (*grau kursiv*).

5.5.5.3 Konsolidierung des Dokumentenschemas (K2.3)

Die Spezifikation von Struktur und Format der Ressourcen-Repräsentationen ist Gegenstand des dritten Entwicklungsschrittes zur Konsolidierung des REST-Architekturmodells (*Schritt K2.3*). Den Ausgangspunkt bilden das initial abgeleitete Dokumentenschema und die Anforderungen an die Dokumentengestaltung, die sich aus der Konsolidierung von OS-ED, OS-VD sowie dem Ressourcenschema ergeben. Als Quelle für die Identifikation und Spezifikation von *Dokumenttypen* (DT) werden die Repräsentationen von Entitätsdiensten, die eingehenden und/oder ausgehenden Nachrichten beim Aufruf von Vorgangsdiensten sowie, optional, die Repräsentationen der Vorgangsdienste selbst herangezogen.

Das Ergebnis des Entwicklungsschrittes ist das *konsolidierte Dokumentenschema*, das alle Stammdaten- oder Transaktionsdaten-Dokumenttypen erfasst, die von Entitäts- und Vorgangsdiensten der RESTful SOA gesendet oder empfangen werden. Zur vollständigen Beschreibung der einheitlichen REST-Schnittstelle werden die spezifizierten DTs sowohl den Ressourcen im Ressourcenschema als auch den Nachrichten im OS-VD zugeordnet.

Die **Konsolidierungsschritte des initialen Dokumentenschemas (K2.3)** sind:

1. Entfernen der Dokumenttypen von Diensten, die als Ergebnis der Konsolidierung von OS-ED oder OS-VD keine Ressourcenzustände veröffentlichen.
2. Hinzufügen von neuen Dokumenttypen für jede neue Entitäts- oder Vorgangsressource.
3. Spezifizieren der Dokumenttypen in Abstimmung mit den Schritten zur Gestaltung der Repräsentationen von Entitätsressourcen und Vorgangsressourcen (vgl. K2.1 (Schritt 4 und 5) und K2.2.1 (Schritt 6, 7 und 13)).
4. Formatieren der Dokumentstruktur und Typisieren der Attribute für die Dokumentformate für die Dokumenttypen.
5. Spezifizieren sowie zuordnen von Dokumenttypen als IN-/OUT-Parameter von HTTP-Methoden der Ressourcen sowie zu Nachrichten der Interaktionsbeziehungen zwischen den Komponenten im OS-VD.

Eine wichtige Aufgabe in K2.3 ist die Abstimmung des initialen Dokumentenschemas mit den Ergebnissen der Konsolidierung von OS-ED, OS-VD sowie des Ressourcenschemas. Beispielsweise sind diejenigen initial abgeleiteten DTs zu entfernen, die mit Diensten korrespondieren, die im Rahmen der Konsolidierung entfernt oder zusammengefasst wurden (Schritt 1).

Das Festlegen von DTs sowie der akzeptierten Dokumentformate (Medientypen) als Parameter der öffentlichen Methoden von RESTful Services beschreibt das Ressourcenschema. Die Beziehung zu den ausgetauschten Nachrichten zwischen Diensten erfasst das OS-VD.

5.5.5.4 Ableitung und Spezifikation des Datenschemas (K2.4)

Das Ziel des Schrittes *Konsolidierung K2.4* ist die Spezifikation des Datenschemas, welches die Strukturinformationen für die Entwicklung des (persistenten) Datenhaltungsteils des Anwendungssystems kapselt. Zusammen mit den Objekttypen von OS-ED und OS-VD, die die Bestandteile von Anwendungs- und Kommunikationsteil des AwS beschreiben, erfolgt im letzten Schritt die Spezifikation der Innensicht der RESTful SOA. Die Quelle für die Ableitung des Datenschemas bilden die zu persistierenden EOTs des konsolidierten OS-ED, deren Zustände die Stamm- und (persistenten) Transaktionsdaten des unterstützten Geschäftsprozesses repräsentieren und daher durch die Datenverwaltungskomponente des AwS dauerhaft zu speichern sind. Das Datenschema wird dabei zunächst als Projektion auf das OS-ED gebildet.

Die Vorgabe von konkreten Entwicklungsschritten ist in K2.4 stets von dem Datenmodell abhängig, das dem Datenmanagementsystem (DMS) des AwS zugrunde liegt. Aus diesem Grund wird das Vorgehen zur Definition des Datenschemas in der SOM-R-Methodik nur in allgemeiner Form beschrieben. Die konkrete Ausgestaltung und Anwendung des Entwicklungsschrittes K2.4 wird nachfolgend am Beispiel der Fallstudie demonstriert.

Die **Definition des Datenschemas** umfasst folgende Entwicklungsschritte (K2.4):

1. Ableiten von Datenobjekttypen (DOT) für die persistent zu verwaltenden EOTs im OS-ED (entsprechend der Markierung „p“).

2. Bestimmen der Eigenschaften (z. B. Attribute und Wertebereiche) von DOTs auf Basis der Eigenschaften (z. B. Attributmenge) der jeweiligen EOTs.
3. Zuordnen eindeutiger Identifikatoren zu jedem DOT.
4. Spezifizieren der Beziehungen zwischen den DOTs entsprechend der Beziehungen zwischen den persistenten EOTs des OS-ED (vgl. Tabelle 5.14).

Hinweis: Die Überarbeitungsschritte von K2.4 fokussieren die Ableitung eines Datenschemas ausgehend vom OS-ED. Weitere Anforderungen können sich darüber hinaus aus den Dokumenttypdefinitionen ergeben. Die Struktur des Datenschemas und die Eigenschaften der DOTs sind deshalb mit diesen abzustimmen.

BEISPIEL FLUGBUCHUNG (KONSOLIDIERTES DATEN- UND DOKUMENTENSCHEMA)

Die konsolidierten Schemata OS-ED und OS-VD bilden die Basis für die Überarbeitung des initialen Dokumentenschemas in Schritt K2.3. Das konsolidierte Dokumentenschema des betrachteten Ausschnitts der Fallstudie zeigt Tabelle 5.13.

Dokumenttyp (p/t)	**Dokumentstruktur (JSON-Format)**
Kunde (p)	"KundeID" : "number (int)", "Nachname" : "string", "Vorname" : "string", "Kundenadresse" : {"PLZ" : "string", "Ort" : "string", "Nr" : "string" }, …
Flug (Sitzplatz) (p)	"FlugID" : "number (int)", "FlugNr" : "string", "Start-/Ankunftszeit" : "string", "Start-/ZielflughafenID": "string", "references" : [{"type":"self", "uri":"/flug/{id}"}, {"type":"flugplan", "uri":"/flugplan/{id}"}], ..., "sitzplatzliste" : [{"PlatzNr" : "string", "Klasse" : "string", "references" : [{"type":"self", "uri":".../sitzplatz/{id}"}], ...]
Suchanfrage (t)	"Hinflug-/Rückflugdatum" : "string", "Hin-/Rückflugzeit" : "string", "Start-/Zielflughafen" : "string", …
Flugstatus (t)	Dokumenttyp: Suchanfrage
Flugdaten (t)	Liste mit verfügbaren Flügen und Sitzplätzen (siehe Dokumenttyp Flug), interne Spezifikationen für Preisberechnung
Suchergebnis (t)	Liste mit verfügbaren Flügen (siehe Dokumenttyp Flug) + je Sitzplatz "Preis" : "number (int)"
Flugwahl (Auftrag) (p)	"FlugID" : "number (int)", "Passagieranzahl" : "number (int)", "Status" : "Flug_gewählt", …
Angebot (Auftrag) (p)	"AuftragskopfID" : "number (int)", "FlugID" : int frac, "Sitzplatz_frei" : [Liste mit "PlatzNr" : "string"], "Status" : "Angebot_erstellt", "references" : [{"type":"self", "uri":"/auftrag/{id}"}, {"type":"kunde", "uri":"/kunde/{id}"}, …]
Angebotsbestätigung (Auftrag) (p)	"AuftragskopfID" : "number (int)", "FlugID" : "number (int)", "Status" : "Angebot_bestätigt", "Sitzplatz_gewählt" : [Liste mit "PlatzNr" : "string"], "Passagierdaten" : [{"Nachname" : "string", "Vorname" : "string", "Passagieradresse": { … }, …], …

Tabelle 5.13: Konsolidiertes Dokumentenschema der Fallstudie (Ausschnitt)

Da aus der Überarbeitung von OS-ED und OS-VD für den betrachteten Ausschnitt kein Änderungsbedarf resultiert, wird keiner der bestehenden DTs entfernt oder ein neuer hinzugefügt (K2.3, Schritte 1 und 2). Auf Basis der Konsolidierungsentscheidungen in K2.1 und K2.2.1 wird die Struktur der DTs verfeinert (Schritt 3). Beispielsweise beschreiben Referenzen ("references") in der Dokumentstruktur die Beziehungen zwischen Objekttypen. So verknüpft die "reference" von *Flug* den gültigen *Flugplan*. Die Sitzplätze eines Fluges sind in einer eigenen "sitzplatzliste" erfasst. Anschließend erfolgt die Formatierung der Dokumenttypen als

JSON und die Typisierung der Attribute (Schritt 4). Die Zuordnung der DTs als Schnittstellenparameter der RESTful Services wird bereits in den Ausführungen zur Konsolidierung von OS-ED und OS-VD dargestellt (Schritt 5).

In der SOM-R-Methodik wird die **Ableitung von Datenschemata** im Rahmen der Fallstudie am *Beispiel des Relationenmodells* demonstriert (Abschnitt 5.3.5). Als datenorientierter Modellierungsansatz wird hierbei das SERM eingesetzt, da das Verständnis, das dem SERM zugrunde liegt, die Modellierung von Existenzabhängigkeiten zwischen DOTs betont (siehe Metamodell in Anhang B.1). Eine ausführliche Erläuterung der Modellbildung mit SERM findet sich in [FeSi13, S. 160 ff.]. Die Abbildung der Metaobjekte des OS-ED auf das SERM fasst Tabelle 5.14 zusammen [FeSi13, S. 224-226].

OS-ED-Metamodell	**SERM-Metamodell**
Entitätsobjekttyp (p)	Datenobjekttyp (E-,ER-,R-Typ)
Attribut EOT	Attribut DOT
interacts_with-Beziehung	Beziehung
is_a-Beziehung	Generalisierung
is_part_of-Beziehung	Beziehung
Kardinalität (Beziehung)	Kardinalität (Beziehung)

Tabelle 5.14: Abbildungsbeziehungen zwischen den Metamodellen von OS-ED und SERM

Die zu persistierenden EOTs werden in je einen DOT überführt und ihre Attribute übernommen. Aufgrund der expliziten Modellierung von Existenzabhängigkeiten im SERM ist eine Transformation der Interaktions- und Aggregationsbeziehungen des OS-ED in korrespondierende Beziehungen zwischen DOTs möglich. Die Kardinalitäten der Beziehungen im OS-ED werden direkt ins SERM übernommen. Eine Spezialisierung der DOTs in E-, ER sowie den zugehörigen R-Typen ergibt sich aus den Existenzabhängigkeiten, die durch Beziehungen zwischen diesen dargestellt sind. Die Kompatibilität der Beziehungen in SOM und SERM wird in [FeSi13, S. 226] tabellarisch dargestellt und erläutert. Zur Abbildung referenzieller Integritätsbedingungen sind die Fremdschlüssel in DOTs sowie die Art der Schlüsselvererbung zu bestimmen [FeSi13, S. 163-165].

Die Struktur der persistenten Datenhaltungsebene, die in Form eines SERM-Datenschemas modelliert wird, ist in der oberen Hälfte der Abbildung 5.17 dargestellt. Darunter findet sich das korrespondierende logische Datenbankschema, welches die Eigenschaften der DOTs beschreibt. Ausgehend von den EOTs des konsolidierten OS-ED werden die DOTs des SERM abgeleitet (K2.4, Schritt 1). Hiervon ausgenommen ist lediglich der Einstiegspunkt *Vertrieb*.

Auf Basis von Attributen der EOTs werden die Attributmengen und Datentypen der DOTs spezifiziert und ihnen Primärschlüssel zugeordnet (Schritte 2 und 3). Für die Realisierung der benutzerdefinierten Datentypen der Fallstudie (z. B. *Adresse* und *Auftragsstatus*) wird angenommen, dass diese durch das eingesetzte Datenbanksystem unterstützt werden (z. B. [KeEi13, S. 433 ff.], [Voss08, S. 310 ff.]).

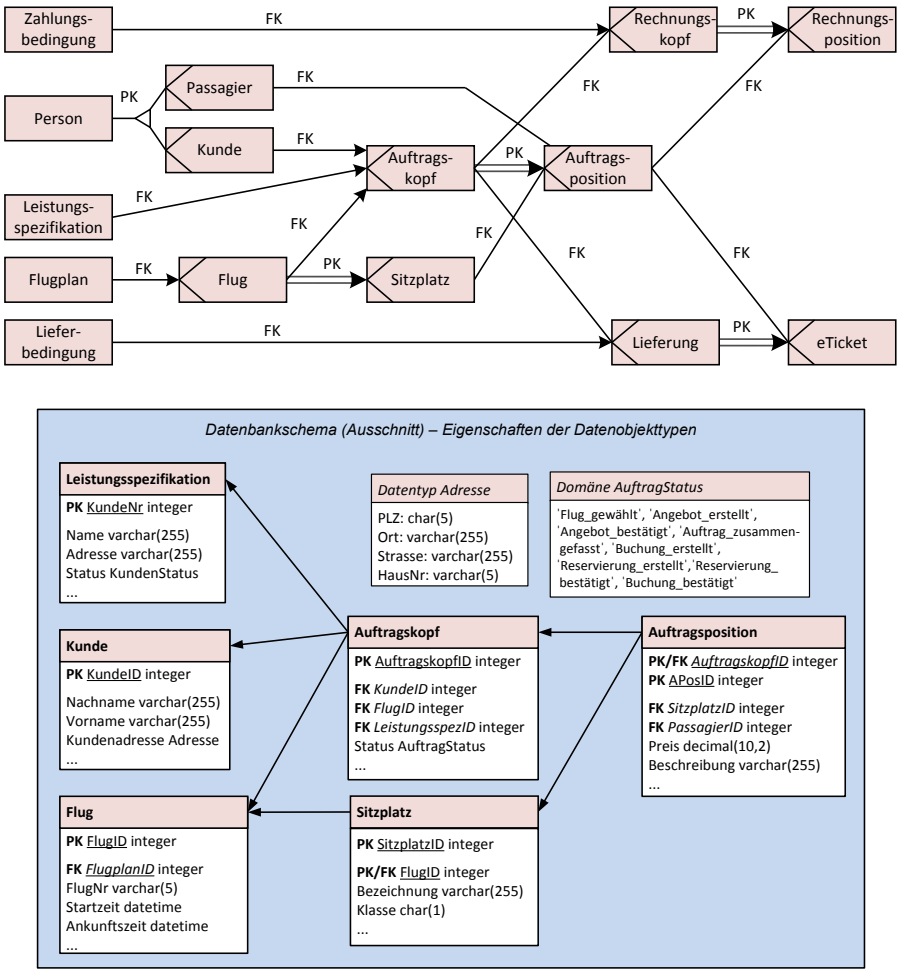


Abbildung 5.17: SERM-Datenschema und Ausschnitt des Datenbankschemas (Fallstudie)

Die interacts_with-Beziehungen zwischen EOTs werden in (0,1)- oder (0,*)-Beziehungen zwischen DOTs überführt und diese entsprechend ihrer existenziellen Abhängigkeiten von links nach rechts angeordnet. Die is_part_of-Beziehungen von *Flug*, *Auftrag*, *Rechnung* und *Lieferung* werden als (1,*)-Beziehungen und der existenziell abhängige „Teil“ als schwacher Entity-Typ modelliert (Vererbungsart PK) [FeSi13, S. 148]. Die Aggregation zwischen *Flugplan* und *Flug* wird (gemäß der Beziehung im OS-ED) als einseitige Abhängigkeit interpretiert und deshalb in einer (0,*)-Beziehung erfasst. Die Generalisierung von *Kunde* und *Passagier* zu *Person* wird ebenfalls in das SERM übernommen. Abschließend werden die Beziehungen zwischen den DOTs in Form von Fremdschlüsselreferenzen realisiert (Schritt 4).

5.5.6 Dritte Transformation (T3): Ableitung des Implementierungsmodells

Gegenstand von Entwicklungsschritt *Transformation T3* ist die automatisierte Erzeugung des Programmcodes einer plattformspezifischen Implementierung der RESTful SOA nach dem Prinzip der modellgetriebenen Softwareentwicklung (MDSD). Den Input (Quellmodell) der Transformation bilden die konsolidierten Schemata des REST-Architekturmodells, aus denen die Schemata der Implementierungsebene abgeleitet werden (Zielmodell). Da T3 die Generierung von Programmcode zum Ziel hat, wird der Schritt meist in Form einer *Modell-zu-Code-Transformation* realisiert (Abschnitt 2.3.1).

Zielsetzung	Generierung von Implementierungsartefakten der RESTful SOA (T3)
Input	Konsolidiertes OS-ED; Konsolidiertes OS-VD; Konsolidiertes Ressourcenschema; Konsolidiertes Dokumentenschema; Strukturdefinition Datenschema
Output	Vorgangsdienste (Programmcode[40]); Entitätsdienste (Programmcode); Schnittstellenspezifikation; Datenbankschema (Strukturdefinition)
Lösungsansatz	Schema-Transformation zur Generierung der plattformspezifischen Implementierung (M2C-Transformation). *Hilfsmittel*: Ansätze und Techniken der modellgetriebenen Softwareentwicklung.

Tabelle 5.15: Zusammenfassung von Entwicklungsschritt *Transformation T3*

Die Ableitung der *Programmartefakte von Entitäts-* und *Vorgangsdiensten* der RESTful SOA erfolgt auf Basis von OS-ED, OS-VD und Ressourcenschema. Die veröffentlichte Schnittstelle der RESTful Services kann dabei aus dem Ressourcenschema in ein Standardformat zur Spezifikation der *REST-Schnittstelle* überführt werden. Ausgehend vom konzeptuellen Datenschema wird für die Beschreibung der Datenbasis eine *Definition des Datenbankschemas* generiert. Die Dokumenttypdefinitionen des Dokumentenschemas fließen in die Implementierung der RESTful Services ein und unterstützen die Gestaltung der Ressourcen-Repräsentationen. Die Durchführung des Entwicklungsschrittes *Transformation T3* entspricht dem automatisierbaren Teil der Aktivität *Realisierung* im Vorgehensmodell der Methodik (Abschnitt 5.4).

Voraussetzung für eine konkrete Ausgestaltung (Instanziierung) des Entwicklungsschrittes ist die Festlegung der Basismaschine sowie der Spezifikationen (Programmiersprachen) zur Implementierung der Zielarchitektur. Der Modellierung des REST-Architekturmodells liegt in dieser Arbeit das objektorientierte Paradigma zugrunde. Grundsätzlich kann das Modell der RESTful SOA somit in die Spezifikationen verschiedenartiger (objektorientierter) Programmierplattformen transformiert werden. Die Konzeption von Transformation T3 erfordert ferner die Erstellung der plattformspezifischen Metamodelle auf Ebene des Implementierungsmodells. Auf dieser Basis wird die metamodellbasierte Abbildungsbeziehung zwischen REST-Architekturmodell und Implementierung definiert.

Den konzeptionellen Aufbau der Transformation T3 visualisiert Abbildung 5.18 in der Form eines MDA-Patterns [OMG03, S. 2-7]. Die Darstellung der Transformation beschränkt sich auf die allgemeine Vorgehensweise (links) [FeSi13, S. 235] sowie die schematische Darstellung des Einsatzes mehrerer Zielplattformen (rechts).

Die Transformation selbst wird als Abbildung der Metaobjekte des Metamodells der REST-Architekturebene auf die Metaobjekte des plattformspezifischen Metamodells der Implementierungsebene definiert. Darüber hinaus können *weitere Informationen* und Techniken zur detaillierten Ausgestaltung des Transformationsprozesses genutzt werden. Die automatisierte

[40] Der Begriff Programmcode umfasst auch Ablaufspezifikationen von WfMS (Workflow-Engine).

Durchführung der Transformation erfolgt meist werkzeuggestützt auf Basis von *Code-generatoren* (z. B. [SVE+07, S. 139 ff.]).

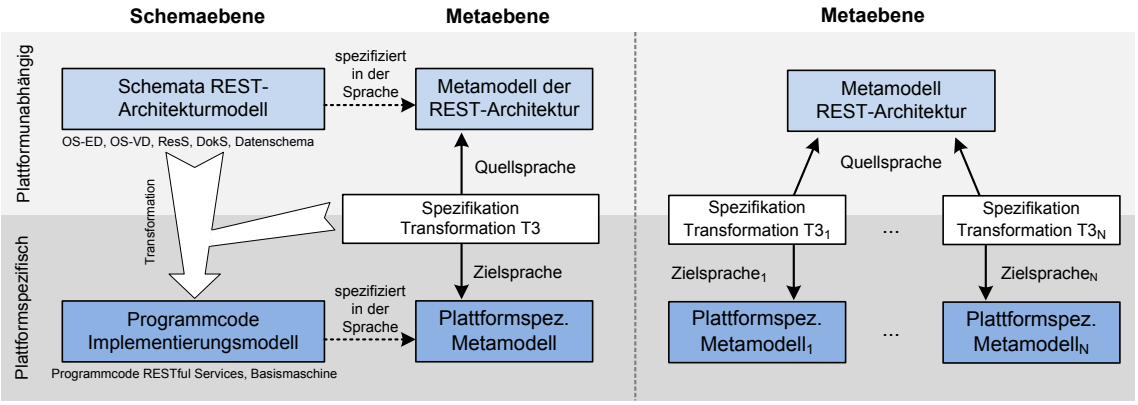


Abbildung 5.18: Allgemeiner einstufiger Aufbau von *Transformation T3*

Zur Unterstützung der Komplexitätsbewältigung, die mit einer Überbrückung der Abstraktionslücke zwischen REST-Architekturmodell und Implementierung einhergeht, ist auch ein mehrstufiger Aufbau der Transformation T3 denkbar (z. B. [PeMe06, S. 125], [OMG03, S. 6-1 ff.]). Nach diesem Prinzip wird der Programmiercode der Implementierung aus einem Zwischenmodell erzeugt, welches zunächst z. B. das plattformspezifische Komponentendesign der Zielarchitektur (*Modell-zu-Modell*) beschreibt. Ist das abgeleitete Zwischenmodell hinreichend detailliert spezifiziert, so wird in einem zweistufigen Ansatz der Quellcode der Softwareartefakte aus diesem generiert (*Modell-zu-Code*). Ein solches Vorgehen ermöglicht die Reduzierung der semantischen Lücke, die in der letzten Transformation zu überbrücken ist.

Als Beispiel einer Transformationstechnik seien an dieser Stelle *Templates* genannt, die häufig zur automatisierten Erzeugung von Quellcode aus formalen Modellen verwendet werden (z. B. [SVE+07, S. 146 ff.], [PeMe06, S. 131 f.], Abschnitt 2.3.2). Aktuell verbreitete Werkzeuge zur softwaretechnischen Unterstützung von M2M- und M2C-Transformationen sind das *Eclipse Modeling Framework* (EMF)[41] der Entwicklungsumgebung Eclipse[42] (z. B. [Gron09, S. 8 f.]) oder die Meta-Modellierungsplattform *ADOxx*[43] (z. B. [BoSi13]). Die Implementierung eines prototypischen Entwicklungswerkzeugs für die SOM-R-Methodik erfolgt in Kapitel 7.

Wie bereits dargelegt, zielt die dritte Transformation auf die automatisierte Erzeugung eines lauffähigen Softwareprogrammes zur Realisierung der RESTful SOA. Im Allgemeinen ist es jedoch nicht möglich alle notwendigen Informationen für eine vollständige Systemerzeugung in der Transformationsspezifikation zu kapseln. In diesen Fällen ist eine manuelle Bearbeitung des generierten Implementierungsmodells (insbes. des Programmcodes) erforderlich, um die nicht-generierbaren Modellelemente um fehlende Informationen anzureichern ([PeMe06, S. 135 ff.], [SVE+07, S. 13]). Die abschließenden Schritte zur Realisierung eines lauffähigen Programms sind Teil von *Konsolidierung K3*.

---

[41] Offizielle Webseite *Eclipse Modeling Framework*: http://www.eclipse.org/modeling/emf/ (Abruf am 25. März 2015)

[42] Webseite *Eclipse*: http://www.eclipse.org/ (Abruf am 25. März 2015)

[43] Webseite *ADOxx*: http://www.adoxx.org/live/home (Abruf am 25. März 2015)

5.5.7 Konsolidierung des Implementierungsmodells (K3)

Das Ziel des Entwicklungsschrittes *Konsolidierung K3* ist die Anreicherung der automatisiert erzeugten Artefakte der RESTful SOA um die nicht-generierbaren Implementierungsinformationen. K3 entfällt, wenn im Zuge des dritten Transformationsschrittes (T3) alle Implementierungsdetails generiert werden. Die Durchführung des Entwicklungsschrittes entspricht dem nicht-automatisierbaren Teil der Aktivität *Realisierung* im Vorgehensmodell der Methodik (Abschnitt 5.4). Mit Abschluss der Konsolidierung liegt die RESTful SOA als lauffähiges AwS vor.

Zielsetzung	Realisierung des Anwendungssystems RESTful SOA (K3)
Input (Generierte Artefakte)	Vorgangsdienste (Programmcode); Entitätsdienste (Programmcode); Schnittstellenspezifikation; Datenbankschema (Strukturdefinition)
Output (Lauffähige Artefakte)	Vorgangsdienste (Programmcode); Entitätsdienste (Programmcode); Schnittstellenspezifikation; Datenbankschema (Strukturdefinition); Konfiguration der Basismaschine
Lösungsansatz	Plattformspezifische Konsolidierung der Implementierungsartefakte für die spezifizierte Zielarchitektur (nicht-automatisiert)

Tabelle 5.16: Zusammenfassung von Entwicklungsschritt *Konsolidierung K3*

Die konkrete Gestaltung des Entwicklungsschrittes K3 setzt die Kenntnis der Transformationsspezifikation und der verwendeten Werkzeuge und Technologien voraus. Deshalb wird an dieser Stelle kein allgemeingültiges Vorgehen für die Durchführung von K3 angegeben. Exemplarisch seien die Notwendigkeit zur manuellen Erstellung von Lösungsverfahren (Methodenrümpfe) nach der Generierung und die Durchführung von Konfigurationsarbeiten an den zugrunde liegenden Basismaschinen genannt (z. B. [FrRu07, S. 39 f.], [PeMe06, S. 135 ff.]).

Der konkrete Entwurf und die Anwendung von Transformation T3 sowie von Konsolidierung K3 werden im nachfolgenden Abschnitt 5.5.8 am Beispiel der plattformspezifischen Implementierung von RESTful Services der Fallstudie für Java und konkrete Basismaschinen gezeigt. In Anhang B.3 werden zusätzlich die Konzeption eines Rails-Metamodells sowie die Spezifikation der Transformation T3 zur Ableitung von Rails-basierten Services aus dem REST-Architekturmodell eingeführt.

5.5.8 Entwicklung des plattformspezifischen Implementierungsmodells am Beispiel der Fallstudie

Die Durchführbarkeit der Entwicklungsschritte T3 und K3 wird nachfolgend anhand der Fallstudie demonstriert. Voraussetzung für die konkrete Ausgestaltung von Transformation und Konsolidierung der Systemimplementierung ist die Festlegung der Basismaschine, die zur Realisierung der RESTful SOA eingesetzt wird (z. B. Programmierplattform und Datenbanksystem). Die Merkmale der Basismaschine bestimmen die Struktur sowie die Spezifikationsdetails des zu erzeugenden Implementierungsmodells.

In der SOM-R-Methodik geht mit dem Übergang auf die Implementierungsebene allgemein auch der Wechsel von einer grafischen (modellbasierten) in eine textuelle Repräsentationsform des entwickelten Systems einher. Konform zum Entwicklungsvorgehen der Methodik wird die dritte Transformation in Form von Abbildungsbeziehungen zwischen Metaobjekten definiert. Die Objekte des Implementierungsmodells kapseln somit den Code zur Spezifikation der Softwarekomponenten des AwS.

In der Fallstudie der vorliegenden Arbeit werden die Softwarekomponenten der RESTful SOA auf Basis von Java bzw. dem Framework JAX-RS realisiert. Der Datenhaltung liegt das Relationenmodell zugrunde. Zunächst werden die plattformspezifischen Metamodelle konzipiert und die Abbildungsbeziehungen zwischen REST-Architekturmodell und Implementierungsmodell definiert. Abschließend werden fehlende Details, die manuell an der abgeleiteten Systemspezifikation zu ergänzen sind, erläutert. Der schematische Aufbau des dritten Transformationsschrittes ist bereits in Abbildung 5.18 dargestellt.

Mit dem Framework *Ruby on Rails* wird im Anhang B.3 dieser Arbeit ein zweites plattformspezifisches Metamodell zur Implementierung der RESTful SOA vorgestellt. Die Konzeption von Rails als alternative Zielarchitektur und der Transformation T3, um es aus dem REST-Architekturmodell abzuleiten, betont die Plattformunabhängigkeit des Entwicklungsschrittes.

5.5.8.1 Spezifikation von RESTful Services mit JAX-RS

Die RESTful Services der Fallstudie werden in der Programmiersprache *Java*[44] entwickelt. Ein Java-Framework zur Realisierung von RESTful Web-Services spezifiziert der Standard *JAX-RS*. Den Kern der JAX-RS-Spezifikation bilden Annotationen, welche REST-Eigenschaften (z. B. URI-Pattern, HTTP-Methoden, produzierte und konsumierte Medientypen) an die Methoden von Java-Klassen binden. Aktuell liegt JAX-RS in der Version 2.0 vor und wird in der JSR-339[45] „JAX-RS 2.0: The Java API for RESTful Web Services" definiert ([Burk13, S. 27], [Ulle12, Abschnitt 13.2.2]).

Als Referenzimplementierung der JAX-RS API wird in dieser Arbeit das *Open-Source-Framework Jersey*[46] eingesetzt. Zusammen mit dem HTTP-Server von Java SE, dem Applikationsserver Glassfish[47] sowie PostgreSQL[48] als relationale Datenbank bilden Java bzw. Jersey die Basismaschine der Fallstudie. Weiterführende Informationen zur Erstellung von Anwendungen mit JAX-RS und Jersey finden sich z. B. in [Gula13] und [Burk13].

Abbildung 5.19 zeigt die Abbildungsbeziehungen von Entwicklungsschritt T3. Die Schemata OS-ED, OS-VD sowie die Spezifikationen von Ressourcen und Dokumenten werden in korrespondierende Java-Klassen und deren Eigenschaften überführt. Die Implementierung der RESTful Services wird durch einen einfachen dreigeteilten Aufbau strukturiert. Grundsätzlich werden

[44] Offizielle Webseite: https://www.java.com/ (Abrufe jeweils am 25. März 2015)

[45] JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services (Web: https://jcp.org/en/jsr/detail?id=339)

[46] Webseite: https://jersey.java.net/

[47] Webseite: https://glassfish.java.net/de/

[48] Webseite: http://www.postgresql.org/

ein Dienst bzw. seine Ressourcen durch eine Interface-, Implementierungs- und Entitätsklasse realisiert.

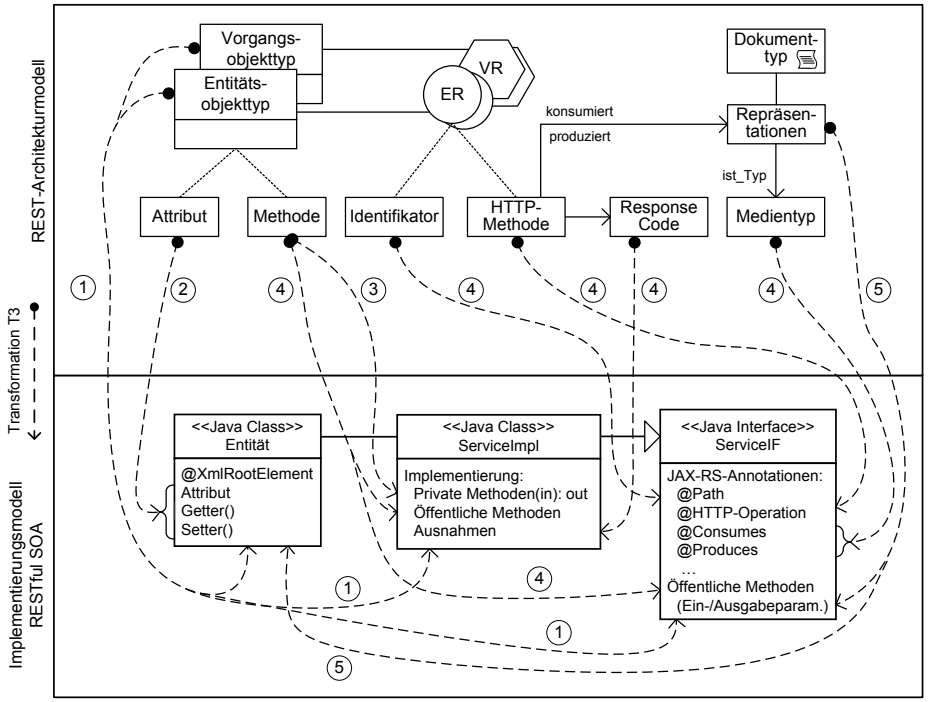


Abbildung 5.19: Abbildungsbeziehungen zwischen den Metamodellen REST-Architekturmodell und JAX-RS-Implementierungsmodell

Das *Service-Interface* (ServiceIF) beschreibt die öffentlichen Methoden und definiert Ressourceneigenschaften in Form von Annotationen. Dabei wird jedem Service bzw. seinen Methoden ein Identifikator/URL (@Path), die implementierten HTTP-Methoden (@GET, @POST, @PUT, @DELETE), verfügbare Medientypen für Request- und Response-Nachrichten (@Produces und @Consumes) sowie optional weitere Parameter von Ressourcenaufrufen (z. B. @QueryParam und @PathParam) zugewiesen. Die Implementierung der Ausführungslogik von Methoden erfolgt in der *ServiceImpl*-Klasse. In den Klassen sind die domänenspezifischen Methodenrümpfe durch den Systementwickler auszuspezifizieren. Im Rahmen der Transformation ist lediglich die Abbildung der BPMN-Lösungsverfahren in eine initiale Version der Methodenfunktionalität möglich (siehe nachfolgende Ausführungen). Die Strukturdefinition von RESTful Services sowie ihrer Ressourcen-Repräsentationen beschreiben die *Entitätsklassen*.

Die Transformation der Bausteine des REST-Architekturmodells in die Bausteine der JAX-RS-Plattform umfasst folgende Schritte:

1. Zu jedem Entitäts- und Vorgangsobjekttypen wird ein Service-Interface, Service-Implementierung sowie eine Entitätsklasse erstellt. Bei Vorgangsdiensten dient die Entitätsklasse der Verwaltung von Zuständen laufender/abgeschlossener Prozessinstanzen.
2. Die Attribute von EOTs, welche persistente Ressourcenzustände beschreiben, werden in Attribute der Entitätsklassen und Zugriffsmethoden überführt.
3. Private Methoden sowie die Input-/Outputparameter und interne Ausnahmen werden der *ServiceImpl*-Klasse zugeordnet.

4. Die Ressourcenschnittstelle wird durch die Klasse *ServiceIF* spezifiziert. Die öffentlichen Methoden der Objekttypen definieren den Aufbau des Interface. Auf Basis des Ressourcenschemas (in Abstimmung mit OS-ED/OS-VD) werden den Methoden Identifikator, HTTP-Methoden sowie Medientypen in Form von Annotationen zugeordnet. Korrespondierend hierzu erfolgt die Spezifikation von öffentlichen Methoden und (öffentlichen) Ausnahmen zur Realisierung der Response Codes in der Klasse *ServiceImpl*.
5. Für Dokumenttypen, welche Ressourcen-Repräsentationen beschreiben, können ebenfalls Entitätsklassen erstellt werden. Zudem werden die Dokumenttypen den öffentlichen Methoden als Parameter zugeordnet.

SPEZIFIKATION DES LÖSUNGSVERFAHRENS

Die Spezifikation der Geschäfts- und Ablauflogik von Diensten erfolgt mit der Workflowsprache BPMN (Abschnitt 5.3.6). Die Transformation von BPMN-Schemata in eine Implementierungsspezifikation wird in der vorliegenden Arbeit nicht näher betrachtet. Eine Realisierung des Lösungsverfahrens erfolgt primär im Zuge des dritten Konsolidierungsschrittes (K3). Für die Erzeugung der Methodenrümpfe ist grundsätzlich die Abbildung der Ausführungslogik von BPMN-Elementen (z. B. Aktivitäten, Ereignissen oder Gateways) auf äquivalente Konstrukte der Programmiersprache zu definieren. Zur Implementierung der Parameter von HTTP-Aufrufen sowie der hierbei ausgetauschten Repräsentationen stehen in Java eine Reihe von REST-Clients zur Verfügung (z. B. [Burk13], [Gula13], [KaMe13]).

Abschließend sei noch auf aktuelle Entwicklungen im Bereich der *Workflow-Management-Systeme* (WfMS) verwiesen. So nutzen beispielsweise Open-Source-WfMS, wie Activiti[49] oder Camunda[50], bereits die BPMN 2.0 für die Spezifikation ihrer Workflows. Zur Verwaltung von definierten Prozessen und Prozessinstanzen stellen diese Systeme eine REST-Schnittstelle zur Verfügung. Ein Workflow ist somit selbst ein RESTful Service, der über seine öffentliche (REST-) Schnittstelle verwaltet wird. HTTP-Aufrufe innerhalb des Workflows sind mittels Java-Code als eigene Aktivität implementierbar. Durch die direkte Ausführung von BPMN-Schemata wird die Abstraktionsebene der Spezifikation weiter angehoben und erlaubt eine „direktere“ Umsetzung des modellierten Lösungsverfahrens. Der Einsatz von WfMS stellt somit eine weitere alternative Technologie zur Implementierung von Vorgangsservices der RESTful SOA dar.

ERSTELLUNG DER DATENBASIS

Das Datenschema des REST-Architekturmodells definiert die Struktur der persistenten Datenhaltung (Abschnitt 5.5.5.4). Da es sich bei der Ableitung von Datenbankschemata um ein standardisiertes Vorgehen handelt, wird sie an dieser Stelle nur in kompakter Form vorgestellt. Die Transformation eines SERM-Schemas in SQL-Statements basiert auf den Ausführungen von SINZ [Sinz14, Kapitel 5.3, S.23 ff.]:

- Zu jedem *Datenobjekttyp* des SERM (E-, ER und R-Typ) wird ein *Relationstyp* (Tabelle) erstellt. Attribute und Datentypen der Datenobjekttypen spezifizieren die Struktur der

[49] Webseite: http://activiti.org/ (Aufrufe jeweils am 25. März 2015)

[50] Webseite: http://camunda.com/de/

Tabelle (SQL: `CREATE TABLE „DOT-Name“ („Attribut“ „Datentyp“, […]);`). Als *Primärschlüssel* werden ein oder mehrere natürliche Attribute bestimmt oder ein künstliches Schlüsselattribut erstellt.

- Die gerichteten Beziehungen zwischen den DOTs werden in jeweils einer *Fremdschlüssel-Definition* abgebildet. Dabei wird der Primärschlüssel jedes Relationstyps von „links nach rechts“ vererbt, so dass (Einträge) rechts stehender Relationstypen stets die Einträge der links stehenden Relationstypen referenzieren und somit von diesen existenzabhängig ist.

Weiterführende Informationen zur Abbildung von Datenschemata auf die Sprache SQL und die Realisierung von Existenzabhängigkeiten zwischen Relationstypen finden sich in der einschlägigen Datenbankliteratur (z. B. [KeEi13, S. 75 ff.], [Voss08, S. 104 ff.]).

Zur Realisierung der Datenhaltungskomponente der RESTful SOA seien exemplarisch noch zwei Technologiealternativen genannt, die in der vorliegenden Arbeit nicht behandelt werden. So werden bei der Überwindung der semantischen Lücke zwischen objektorientierter Programmiersprache und relationaler Datenbank häufig Softwareframeworks für das *objekt-/relationale Mapping* (O/R-Mapping), sogenannte O/R-Mapper[51], eingesetzt (z. B. [MüWe12], [Russ08]). Die zweite Implementierungsalternative stellen *Dokumentdatenbanken* dar, die eine direkte Speicherung von ausgetauschten Dokumenten ermöglichen, ohne dass im AwS eine Abbildung auf das Datenmodell des zugrundliegenden DBS erforderlich ist (z. B. [LaCr13, S. 120 ff.]). Die Nutzung von Dokumentdatenbanken bietet sich gerade im Hinblick auf den dokumentgetriebenen Nachrichtenaustausch zwischen RESTful Services an [WoBe13, S. 2].

Spezifikation der Mensch-Computer-Kommunikation

Die Eigenschaften der (R-)IOTs beschreiben die Merkmale und Funktionalität der Kommunikationsschnittstelle zu den externen Komponenten der RESTful SOA (Abschnitt 5.5.5.2). Zusammen mit der Spezifikation der ein- und ausgehenden Nachrichten sind R-IOTs der Ausgangspunkt für die Gestaltung und Implementierung der Komponenten der Nutzeroberfläche bzw. Schnittstelle für die MCK- oder CCK-Kommunikation. Als typische Realisierungsformen der MCK-Schnittstelle seien client- und serverseitige Web-Anwendungen, Webseiten sowie Emails genannt. Der Aufbau und die Bestandteile der auszutauschenden Nachrichten sowie die Definition von Verknüpfungen zwischen RESTful Services werden auf Basis der (zu übermittelnden) Dokumenttypen spezifiziert, die den R-IOTs über ihre Beziehung zu Vorgangsdiensten zugeordnet sind. Die Abbildungsbeziehungen zwischen R-IOTs und den Komponenten der MCK-Schnittstelle zeigt Abbildung 5.20.

Die Kapselung von Funktionalität der externen Nutzerschnittstellen in R-IOTs unterstützt die Erzeugung von Clientanwendungen. (1) Jeder R-IOT wird in eine externe Nutzerkomponente überführt (z. B. Webseite). (2) Die bereitgestellte Funktionalität der Methoden des R-IOT kann direkt durch die Nutzerkomponente realisiert, oder auch (teilweise) auf die Klasse des

[51] Softwareframeworks für O/R-Mapping sind z. B. *Java Persistence API* (*JPA*) (http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html, Abruf am 11. Januar 2015) und *Hibernate* (http://hibernate.org/, Abruf am 11. Januar 2015).

zugeordneten VOT ausgelagert werden. Die im Rahmen der Nutzerinteraktion zu verarbeitenden Daten sowie der Aufbau der zu sendenden bzw. zu empfangenden Nachrichten bestimmen (2) die Attribute des R-IOT sowie (3) die Dokumenttypen als Parameter der (4) HTTP-Methoden des zugehörigen Vorgangsdienstes.

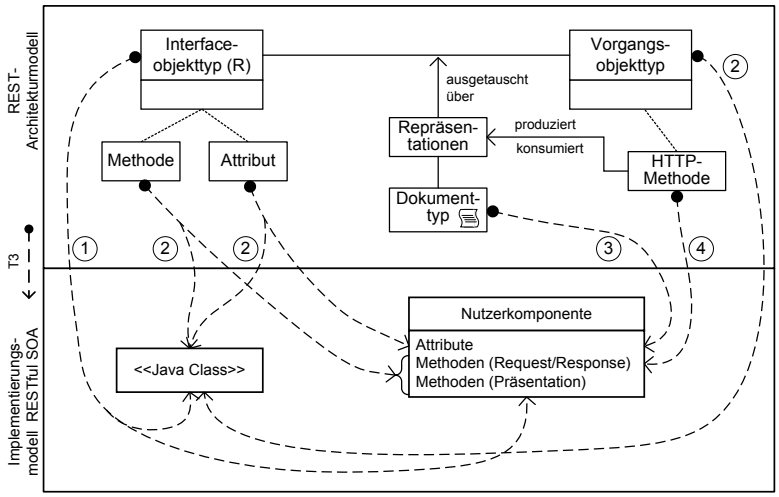


Abbildung 5.20: Abbildungsbeziehungen zwischen dem R-IOT und der MCK-Nutzerschnittstelle

Die Implementierung von Vorgangsdiensten mit Beziehungen zu externen Aufgabenträgern wird in diesem Abschnitt am Beispiel von HTML-Webseiten veranschaulicht. Ein Versand und Empfang von HTTP-Nachrichten ist clientseitig auf Basis von JavaScript realisierbar[52]. Alternativ ist eine Implementierung der Nutzerschnittstelle z. B. mit serverseitigen Scripts oder der Erstellung einer Java-basierten Clientanwendung möglich.

KONFIGURATION VON APPLIKATIONS- UND DATENBANKSERVER

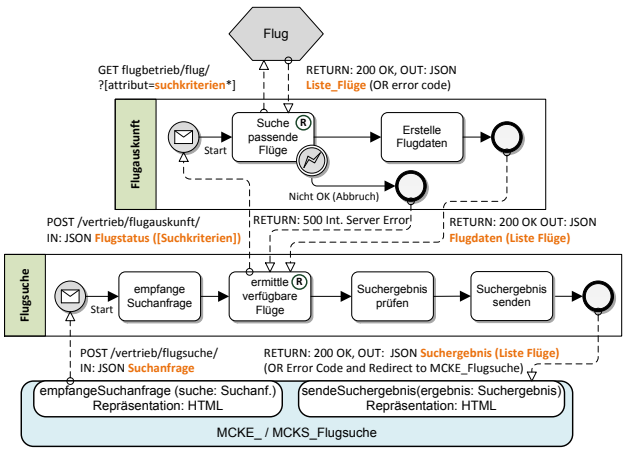
Der Betrieb des AwS erfordert die Konfiguration von Web-, Applikations- und Datenbankserver als Teil des dritten Konsolidierungsschrittes. Für den Datenbankserver sind z. B. die Datenbank zu erstellen und die SQL-Statements zum Anlegen der Datenbankschemata zu übergeben. Dem Webserver und dem Anwendungsserver werden die Basis-URL und die Serviceklassen bekannt gemacht bzw. deployed. Die Administration der Basismaschine wird im Rahmen dieser Arbeit nicht weiter untersucht.

5.5.8.2 Beispielhafte Implementierung von RESTful Services

Die Durchführung der Entwicklungsschritte T3 und K3 zur Implementierung der RESTful SOA werden beispielhaft anhand der Fallstudie demonstriert. Als detailliert zu erläuternder Ausschnitt wird für die genauere Erläuterung die Durchführung der *Flugsuche* im Rahmen des Geschäftsprozesses der Flugbuchungsabwicklung gewählt. Die an der Flugsuche beteiligten Komponenten sind der Entitätsservice *Flug*, der interne Vorgangsservice *Flugauskunft* sowie der Vorgangsservice *Flugsuche* (mit MCK-Schnittstelle).

[52] Die asynchrone Übertragung von Daten zwischen Webseite und Webserver wird im Rahmen von HTTP-Aufrufen häufig mit dem Konzept Ajax (Asynchronous JavaScript and XML) realisiert. Ajax-Anwendungen werden meist mit JavaScript und dem XMLHttpRequest-Objekt implementiert [Carl06].

Abbildung 5.21 zeigt den Ausschnitt des konsolidierten OS-VD, welches das Zusammenwirken der RESTful Services bei der Ermittlung von passenden Flügen zu einer Suchanfrage beschreibt. Der Input von Transformation T3 sind somit die Objekttypen *Flug*, *Flugauskunft* und *Flugsuche* sowie die Spezifikationen der zugehörigen Ressourcen und die Dokumenttypen *Suchanfrage*, *Flugstatus*, *Flugdaten* und *Suchergebnis* aus dem REST-Architekturmodell der Fallstudie (Abbildung 5.16).



Dokumenttyp	Dokumentstruktur (JSON-Format)
Flug (Sitzplatz)	"FlugID" : "number (int)", "FlugNr" : "string", "Start-/Ankunftszeit" : "string", "Start-/Ziel-flughafenID": "string", "references" : [{"type":"self", "uri":"/flug/{id}"}, {"type":"flugplan", "uri":"/flugplan/{id}"}], ... , "sitzplatzliste" : [{"PlatzNr" : "string", "Klasse" : "string", "references" : [{"type":"self", "uri":".../sitzplatz/{id}"}], ...]
Suchanfrage	"Hinflug-/Rückflugdatum" : "string", "Hin-/Rückflugzeit" : "string", "Start-/Zielflughafen" : "string", …
Flugstatus	Dokumenttyp: Suchanfrage
Flugdaten	Liste mit verfügbaren Flügen und Sitzplätzen (siehe Dokumenttyp Flug), interne Spezifikationen für Preisberechnung
Suchergebnis	Liste mit verfügbaren Flügen (siehe Dokumenttyp Flug) + je Sitzplatz "Preis" : "number (int frac)"

Ressourcen	URL	HTTP-Operatoren	Request (IN), Response (OUT)
Flug	/flugplan/{id}/flug/	*GET, POST*	
	/flugplan/{id}/flug/?{attribut=param}+	GET	OUT: Liste Flüge (zu 'param')
	/flugplan/{id}/flug/{id}	*GET, PUT, DELETE*	
Flugsuche	/flugabfrage/	POST	IN: Flugsuche, OUT: Suchergebnis
	/flugabfrage/{id}	*GET, PUT, DELETE*	
Flugauskunft	/flugauskunft/	POST	IN: Flugstatus, OUT: Flugdaten
	/flugauskunft/{id}	*GET, PUT, DELETE*	

Flug
PK FlugID integer
FK *FlugplanID* integer FlugNr varchar(5) Startzeit datetime Ankunftszeit datetime Startflughafen varchar(255) Zielflughafen varchar(255) ...

Abbildung 5.21: Konsolidiertes OS-VD der Fallstudie (Ausschnitt)

Aus dem EOT *Flug* werden das Interface *FlugServiceIF* sowie die Java-Klassen *Flug* und *FlugServiceImpl* erzeugt. Die Schnittstellenspezifikation des Flugservice wird den Methoden des *FlugServiceIF* in Form von Annotationen zugeordnet. Für jede Ressource des Flugservice werden ausgehend vom Ressourcenschema die URL (@Path), HTTP-Methoden (@HTTP-Operation) sowie die produzierten (@Produces) und konsumierten (@Consumes) Repräsentationen bzw. Dokumenttypen und Medientypen als Annotationen den öffentlichen Methoden des Interface hinzugefügt. Die Klasse *FlugServiceImpl* realisiert die Lösungsverfahren der spezifizierten Methoden und verwaltet die persistenten Ressourcenzustände. In der Klasse *Flug* werden die Attribute des Dienstes und zugehörige Getter/Setter definiert, die Basis für die Verwaltung der Zustände und Repräsentationen sind. Nähere Informationen zum Quellcode des implementierten Beispiels können beim Autor der vorliegenden Arbeit angefragt werden.

Die Interfacedefinition von Entitätsdienst *Flug* zeigt Quellcode 5.1. Die Ressource zur Selektion von Flügen anhand von Suchkriterien (`/flug?suche=param`) wird über den Parameter (`@QueryParam`) aufgerufen. Ist dieser leer, so werden alle Flüge zurückgegeben (`/flug/`). Das Ergebnis der Verarbeitung ist eine JSON-Serialisierung der Liste des Objekts *Flug*

(`List<Flug>`). Das Marshalling von Objekten kann dabei automatisiert auf Basis von JAXB-Annotationen[53] und des produzierten Typs der annotierten Methode erfolgen.

```
@Path("/flugbetrieb/flug")
public interface FlugServiceIF {
    @GET
    @Produces( MediaType.APPLICATION_JSON )
    public List<Flug> readAll(@QueryParam("suche") String param);

    @POST
    @Consumes( MediaType.APPLICATION_JSON )
    public Response create(InputStream is);

    @GET
    @Path("/{id}")
    @Produces( MediaType.APPLICATION_JSON )
    public Flug read(@PathParam("id") int id);

    @PUT
    @Path("/{id}")
    @Consumes( MediaType.APPLICATION_JSON )
    public Response update(@PathParam("id") int id, InputStream is);

    @DELETE
    @Path("/{id}")
    public Response delete(@PathParam("id") int id);
}
```

Quellcode 5.1: Java-Interface des Entitätsservices *Flug*

Die Java-Klasse der Entität *Flug* definiert die Attribute sowie Getter- und Setter-Methoden des Objekttyps *Flug*. Die JAXB-Annotation unterstützen die Serialisierung und Deserialisierung von Objekten nach bzw. von JSON (und XML). Für die definierten Dokumenttypen *Suchanfrage*, *Flugstatus*, *Flugdaten* und *Suchergebnis* sind ebenfalls Entitätsklassen zu erstellen.

```
@XmlRootElement
public class Flug {
    @XmlAttribute
    private int id;
    @XmlElement
    private int flugplanId;
    @XmlElement
    private String flugNr;
    @XmlElement
    private String startzeit;
    @XmlElement
    private String ankunftszeit;
    @XmlElement
    private String startFH;

    // weitere Attribute sowie Getter/Setter
}
```

Quellcode 5.2: Java-Klasse der Entität *Flug*

Die Abbildung von Objekttypen und Ressourcen der Vorgangsservices *Flugauskunft* und *Flugsuche* erfolgt ebenfalls auf Basis der Transformationsspezifikation in Abbildung 5.19. Auf die Verwaltung von Prozessinstanzen wird an dieser Stelle verzichtet und es wird keine persistente

[53] Java Architecture for XML Binding (JAXB). Offizielle Webseite der Java Spezifikation JSR-222: https://jcp.org/en/jsr/detail?id=222 (Abruf am 11. Januar 2015)

Datenstruktur abgeleitet. Einen Ausschnitt der Interfaces *FlugauskunftIF* und *FlugsucheIF* zeigt Quellcode 5.3.

```
@Path("/flugbetrieb/flugauskunft")
public interface FlugauskunftRessourceIF {
    @POST
    @Consumes( MediaType.APPLICATION_JSON )
    @Produces( MediaType.APPLICATION_JSON )
    public Response create(InputStream flugstatus);
    // weitere HTTP-Methoden zur Verwaltung der Flugauskunft
}

@Path("/vertrieb/flugsuche")
public interface FlugsucheRessourceIF {
    @POST
    @Consumes( MediaType.APPLICATION_JSON )
    @Produces( MediaType.APPLICATION_JSON )
    public Response create(InputStream suchanfrage);
    // weitere HTTP-Methoden zur Verwaltung der Flugsuche
}
```

Quellcode 5.3: Java-Interface der Vorgangsservices *Flugauskunft* und *Flugsuche*

Von Interesse ist im Zusammenhang mit Vorgangsservices das Starten einer Vorgangsinstanz, die durch POST z. B. auf die Listenressource *Flugauskunft* und die Übergabe des Dokuments *Flugstatus* in der Anfragenachricht erzeugt wird. Als Ergebnis der Anfrageverarbeitung wird ein Responseobjekt (`Response`) zurückgegeben, welches neben dem HTTP-Status auch die ermittelten *Flugdaten* enthält. Der Rückgabeparameter des Vorgangsdienstes *Flugsuche* ist das Responseobjekt *Suchergebnis*, welches eine Liste von Flügen als Ergebnis der empfangenen *Suchanfrage* beinhaltet.

Die Klasse *FlugauskunftImpl* realisiert das BPMN-Lösungsverfahren von Vorgangsservice *Flugauskunft*. Die Durchführung von HTTP-Aufrufen der Vorgangsservices basiert auf der Client-API von Jersey. Quellcode 5.4 zeigt die Realisierung der REST-Aktivität *suche passende Flüge* im Vorgangsservice *Flugauskunft* im betrachteten Ausschnitt der Fallstudie. Dabei wird die GET-Methode der Listenressource *Flug* aufgerufen und ihr das Dokument *Flugstatus* (flugStatusParam) mit Suchparametern übergeben. Im Falle einer fehlerfreien Abfrage von Flügen wird der Status des Responseobjekts auf *200 OK* gesetzt und die ermittelten *Flugdaten* übergeben. Im Falle einer fehlerhaften Verarbeitung lautet der Antwortstatus auf *500 internal server error*.

```
public Response create(InputStream is) {...
  ...
  WebResource flugservice =  Client.create()
             .resource( "http://localhost:8080/flugbuchung/flugbetrieb/" );
    List<Flug> flugdaten = flugservice.path("flug")
             .queryParam("suche", flugStatusParam)
             .accept( MediaType.APPLICATION_JSON ).get( flugdatenList );
  ...
  return Response.status(200).entity(flugdaten).build()
  } catch (Exception e) {
    throw new WebApplicationException(Response.Status.INTERNAL_SERVER_ERROR);
  }
}
```

Quellcode 5.4: GET von Vorgangsressource *Flugauskunft* auf Entitätsressource *Flug*

Neben der Spezifikation der Entitäts- und Vorgangsdienste werden die Definitionen der Datenbankschemata (SERM) sowie die Konfigurationen von HTTP- und Datenbankserver abgeleitet.

Die Strukturdefinitionen zur Erstellung des relationalen Datenbankschemas können aus dem Datenschema des REST-Architekturmodells generiert und der Serverklasse als ausführbare SQL-Statements hinzugefügt werden (`prepareStatement`). Eine automatisierte Ableitung von SQL aus einem SERM-Schema wird in [Sinz14, Kapitel 5.3] erläutert. Die Konfiguration des http-Servers beschränkt sich aufgrund der Integration in Jersey auf die Übergabe der Serviceklassen (z. B. `FlugService.class`) und Basis-URI der RESTful SOA (z. B. *http://localhost:8080/flugbuchung*). Die Klasse zur Initialisierung und Ausführung der Server ist fast vollständig automatisiert erzeugbar. Einen Ausschnitt aus der Server-Klasse und das CREATE-Statement zum ER-Typ *Flug* zeigt Quellcode 5.5.

```
...
dbConnection = DriverManager.getConnection
             ("jdbc: postgresql://localhost/flugbuchung",user,pwd);
   try{
   dbConnection.prepareStatement
          ("CREATE TABLE flug (flugId INTEGER IDENTITY PRIMARY KEY,
          flugplanId INTEGER, flugNr VARCHAR (8), startzeit VARCHAR (20),
          ankunftszeit VARCHAR (20), startFH VARCHAR (255), zielFH VARCHAR
          (255));").execute();
   ... }
ResourceConfig config = new ResourceConfig
                    (FlugService.class, FlugauskunftService.class,…);
HttpServer server = JdkHttpServerFactory.createHttpServer
                                                (BASE_URI,config);
...
```

Quellcode 5.5: Create-Statement des Relationstyps *Flug*

Nicht automatisiert erzeugt werden im vorliegenden Beispiel die Details der Server-Konfiguration wie z. B. Nutzername und Passwort der Datenbank. Des Weiteren sind in den Methodenrümpfen der Entitätsservices die Lösungsverfahren zur Prüfung der übertragenen Ressourcendaten und SQL-Statements für den DB-Zugriff zu implementieren.

Abschließend wird die Gestaltung der Webseiten als Präsentation der personellen Nutzerschnittstelle des R-IOT *MCKE/S_Flugsuche* kurz betrachtet. Abbildung 5.22 zeigt den Grundaufbau der Seiten. Die Struktur des Suchformulars (links) und der Ergebnistabelle (rechts) wird durch die Definitionen der Dokumenttypen *Suchanfrage* bzw. *Suchergebnis* bestimmt (vgl. Tabelle 5.13).

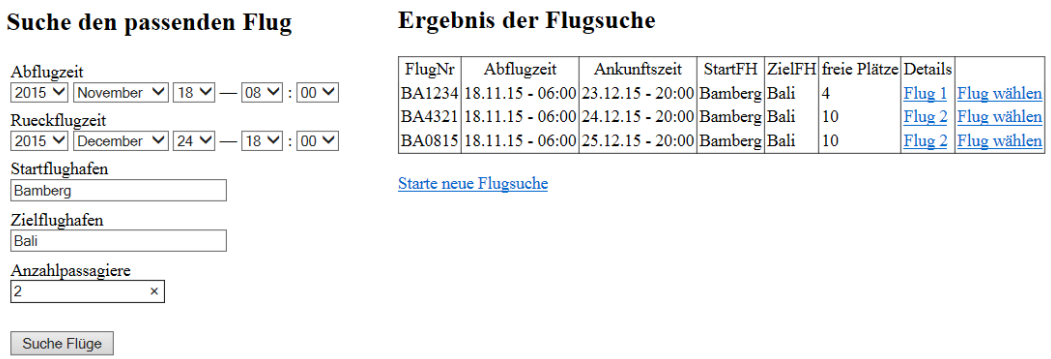


Abbildung 5.22: Grundstruktur der Webseiten zum R-IOT *Flugsuche*

Nach Eingabe der Suchdaten durch den personellen Nutzer können diese über die Schaltfläche „Suche Flüge“ an den Dienst *Flugsuche* übermittelt werden. Die Funktionalität der Webseite

(Implementierung von *MCKE/S_Flugsuche*) übernimmt die Erzeugung des JSON-Dokuments aus den Formulardaten sowie die Realisierung des HTTP-POST an die Listenressource */flugsuche/*. Als Ergebnis der Anfrageverarbeitung wird das Dokument *Suchergebnis* zurückgeliefert, welches eine Liste von Flügen enthält. Über den Link *Details* der einzelnen Zeilen können die Daten der jeweiligen Flugressource aufgerufen werden (`GET \flug\{id}`). Der Link „Flug wählen“ markiert den Aufruf des Dienstes *Flugwahl* und die Übergabe der gewählten FlugID (`POST \flugwahl\`).

5.6 Resümee der konzeptuellen Ergebnisse

Die SOM-R-Methodik des fünften Kapitels ist der Lösungsansatz, um das erste Untersuchungsziel dieser Forschungsarbeit zu erreichen. Mit dieser Methodik wird die modellbasierte Spezifikation von RESTful SOA aus SOM-Geschäftsprozessmodellen in einen einheitlichen Beschreibungsrahmen gefasst und die Durchführung der zu behandelnden Systementwicklungsaufgabe systematisch angeleitet.

In den nachfolgenden Ausführungen werden die Erkenntnisse, die sich im Hinblick auf die Durchführung der Systementwicklung mit der SOM-R-Methodik gewinnen lassen, anhand der Abhängigkeiten und Beziehungen zwischen den Metaobjekten der Modellebenen des SOM-R-Architekturmodells zusammenfassend dargestellt. Der Betrachtungsschwerpunkt liegt dabei auf der Rolle, welche Semantik und Begrifflichkeiten der modellierten GP-Bausteine und ihrer Abbildung auf die Komponenten der RESTful SOA im Entwicklungsprozess spielen (Abschnitt 5.6.1). Anschließend wird das Erreichen der in Abschnitt 1.2.2 festgelegten Formalziele reflektiert (Abschnitt 5.6.2).

5.6.1 Überführung von SOM-Geschäftsprozessmodellen in RESTful SOA

Für die Metaobjekte der SOM-Geschäftsprozessmodellebene wird nun erläutert, wie sich diese in den Komponenten der RESTful SOA wiederfinden.

Betriebliche Transaktion

Betriebliche Transaktionen spezifizieren in SOM-Geschäftsprozessmodellen die Koordination der an der Leistungserstellung und -übergabe beteiligten betrieblichen Objekte in hierarchischer oder nicht-hierarchischer Form [FeSi13, S. 201]. Abbildung 5.23 visualisiert die modellbasierte Abbildung von betrieblichen Transaktionen auf die Komponenten der RESTful SOA mit der SOM-R-Methodik.

Die Operationen zur Überarbeitung der Struktursicht des Informationssystems sind auf der linken Seite zusammengefasst. Die definierten Operationen der Schema-Konsolidierung können in zwei grundlegende Klassen eingeteilt werden:

- Operationen für das Überarbeiten der Schemastruktur mittels Entfernen, Zerlegen oder Zusammenfassen von Bausteinen und Beziehungen zwischen Bausteinen.
- Operationen zur Spezifikation der Eigenschaften von Bausteinen (Anreichern).

Das Hinzufügen von „neuen“ Bausteinen zum Modellsystem erfolgt nur in Ausnahmefällen, um z. B. konsolidierte Schemata aufeinander abzustimmen (z. B. Erstellen einer eigenen Ressource für stornierte Aufträge, Abschnitt 5.5.5.1).

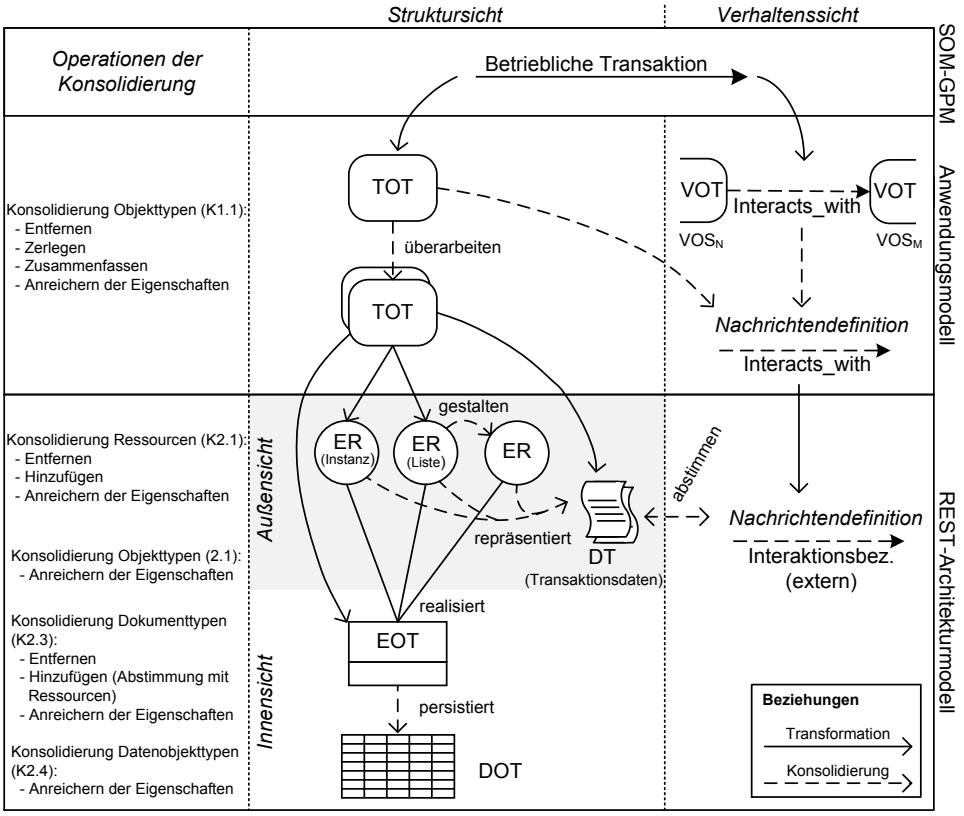


Abbildung 5.23: Beziehungen zwischen der betrieblichen Transaktion und der RESTful SOA

Aus strukturorientierter Sicht (IAS) werden die zu automatisierenden **Transaktionen** des SOM-Geschäftsprozessmodells im Rahmen der modellbasierten Entwicklung zunächst in transaktionsspezifische Objekttypen des Anwendungsmodells (fachliche AwS-Spezifikation) und anschließend, nach Durchführung der Schema-Konsolidierung, in *Entitätsressourcen* sowie *Dokumenttypen* des REST-Architekturmodells transformiert (Außensicht). *Entitätsobjekttypen* spezifizieren die Realisierung von Entitätsressourcen, die zur Persistierung ihrer Zustände auf *Datenobjekttypen* zugreifen (Innensicht). In der verhaltensorientierten Sicht (VES) beschreiben Transaktionen *interacts_with-Beziehungen* zwischen VOTs (Anwendungsmodell) bzw. Vorgangsressourcen (REST-Architekturmodell). Die Definition der Nachrichten, die durch Ressourcen gesendet oder empfangen werden, erfolgt in Abstimmung mit den Spezifikationen von TOTs bzw. den daraus abgeleiteten DTs der Struktursicht.

Die charakteristischen Merkmale von betrieblichen Transaktionen und ihre Entsprechung in der Spezifikation der RESTful SOA werden nachfolgend dargelegt:

- Betriebliche Transaktionen stellen einen *Kommunikationskanal* zwischen zwei betrieblichen Objekten bereit, über den *Lenkungsnachrichten* und *Leistungspakete* übertragen werden. Die ausgetauschten Lenkungsnachrichten und Leistungspakete sind (in Informationssystemen) von der Beziehungsart *Information* [FeSi13, S. 203 f.].

- Die durch Transaktionen übertragenen Informationen (Lenkungsnachricht, Leistungspaket) entsprechen *Ereignissen*, die als Vorereignis die Durchführung der Aufgabe des empfangenden betrieblichen Objekts auslösen, oder als Nachereignis das Ergebnis der Aufgabendurchführung des sendenden betrieblichen Objekts darstellen [FeSi13, S. 203 f.]. Jede Transaktion korrespondiert auf Instanzebene demnach mit einem Ereignis, das einen bestimmten *Ausführungszustand* im konkreten Geschäftsvorfall beschreibt.

In SOM-GPMs übernehmen Transaktionen die Rolle eines integrierenden Bausteins, da sie als einziges Metaobjekt des SOM-Metamodells in Struktur- sowie Verhaltenssicht des Geschäftsprozessmodells erfasst werden (Abbildung 4.5). Die Bedeutung des GP-Bausteins *betriebliche Transaktion* spiegelt sich in den daraus entwickelten Komponenten der RESTful SOA wider:

- *Entitätsressourcen* realisieren Dienste zur Verwaltung der (informationsbasierten) Lenkungsnachrichten und Leistungspakete von betrieblichen Transaktionen. Diese Informationen stellen in der RESTful SOA Transaktionsdaten dar, welche spezifische Ausführungszustände im Ablauf der Geschäftsprozessdurchführung beschreiben. Der Gegenstand transaktionsspezifischer Entitätsressourcen kann z. B. anhand des Transaktionstyps (AVD oder SK) weiter konkretisiert werden.
- *Transaktionsspezifische Dokumenttypen* definieren die Datenstruktur der Informationen, die betriebliche Transaktionen zwischen Aufgaben übertragen. Sie repräsentieren die Zustände von Entitätsressourcen und werden als Nachrichten zwischen Vorgangsressourcen ausgetauscht.
- *Interaktionsbeziehungen* zwischen Vorgangsdiensten werden auf Basis der Transaktionsbeziehungen zwischen Aufgaben im VES erzeugt. Die zwischen den Vorgangsdiensten ausgetauschten *Nachrichten* übertragen Transaktionsdaten in Dokumentform, die den Zustand eines konkreten Geschäftsablaufs spezifizieren.

Beispiel: Zur Verdeutlichung des Sachverhalts werden aus der Fallstudie die betrieblichen Vereinbarungstransaktionen *Anfrage*, *Angebot* und *Buchung* zwischen *Vertrieb* und *Kunde* herangezogen (Abbildung 4.11). Nach Transformation der Transaktionen in Objekttypen und Beziehungen, und ihrer Konsolidierung in KOS bzw. VOS werden die gebildeten KOTs in Entitätsressourcen zu *Auftragskopf* und *-position* sowie in transaktionsspezifische Dokumenttypen als Repräsentation der Ausführungszustände (*Anfrage*, *Angebot* und *Buchung*) der Vereinbarungsphase im Geschäftsprozess überführt (Abbildung 5.15). Die Beziehung zwischen Dokumenttypen und den zwischen Vorgangsdiensten übermittelten Nachrichten (unterschiedlicher OS-VDs) bleibt erhalten (Abbildung 5.16).

AUFGABE UND OBJEKTINTERNES EREIGNIS

Im Allgemeinen werden Geschäftsprozesse aus der verhaltensorientierten Sicht als ereignisgesteuerter Ablauf von Aufgaben beschrieben ([FeSi13, S. 200], [BeKa11, S. 5 f.], [Sche02, S. 3]). Die Überführung der Bausteine *Aufgabe* und *objektinternes Ereignis* des SOM-GPM in die Komponenten der RESTful SOA wird in Abbildung 5.24 gezeigt. Die Operationen zur Überarbeitung

der Verhaltenssicht werden durch den Konsolidierungsschritt der betrieblichen Aufgabe bestimmt und auf der linken Seite der Abbildung dargestellt.

Eine **Aufgabe** wird in der SOM-R-Methodik zunächst in einen VOT auf der Ebene des Anwendungsmodells transformiert, der im Zuge der Konsolidierung überarbeitet und dem, bei Bedarf, ein IOT zugeordnet wird. Anschließend erfolgt die Transformation von VOTs in Vorgangsressourcen, welche die Funktionalität des zu unterstützenden Vorgangs veröffentlichen (Außensicht RESTful SOA). Die Realisierung der Vorgangsressourcen und ihrer externen Kommunikationsbeziehungen erfolgt im REST-Architekturmodell durch eine Menge von VOTs (e/ne) und R-IOTs (Innensicht). Auf Basis der initialen elementaren Vorgangsressourcen (und der zugehörigen VOTs) werden die Service-Koordination gestaltet, weitere Vorgangsressourcen (elementare und/oder nicht-elementar) abgeleitet und, korrespondierend dazu, das Objektschema überarbeitet. DTs können zur Repräsentation der Ausführungszustände von Vorgängen erzeugt werden.

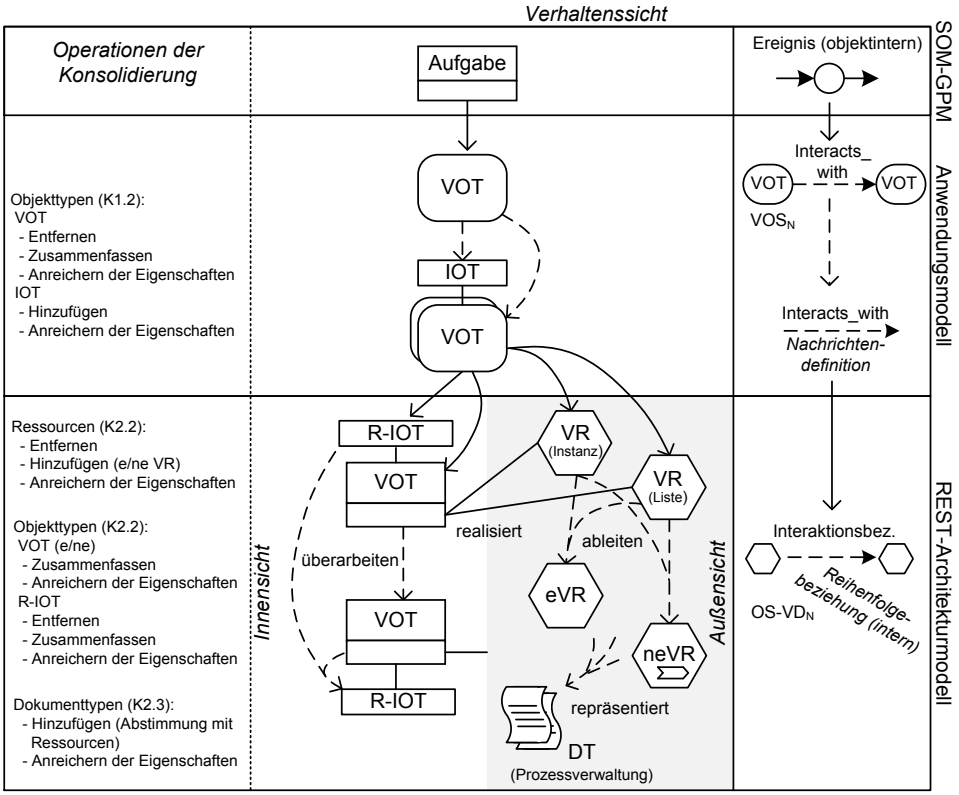


Abbildung 5.24: Beziehungen zwischen Aufgabe und Ereignis (objektintern) sowie der RESTful SOA

Objektinterne Ereignisse spezifizieren interacts_with-Beziehungen, welche die Auslösung von VOTs eines gemeinsamen VOS über Nachrichten beschreiben (Anwendungsmodell) und dadurch den ereignisgesteuerten Ablauf der Aufgabendurchführung abbilden. Im REST-Architekturmodell spezifizieren Interaktionsbeziehungen die logische Ausführungsreihenfolge der Vorgangsdienste eines OS-VD zur Durchführung von Geschäftsvorfällen.

Die charakteristischen Merkmale von Aufgaben und internen Ereignissen, und ihre Entsprechung in der Spezifikation der RESTful SOA werden nachfolgend betrachtet [FeSi13, S. 98 f.]:

- Eine Aufgabe führt als Vorgang ein Lösungsverfahren auf einem Aufgabenobjekt aus und verfolgt dabei Sach- und Formalziele.
- Vorereignisse lösen als Input die Durchführung von Aufgaben aus, die wiederum Nachereignisse produzieren.
- Die gemeinsamen Aufgaben eines betrieblichen Objekts werden über objektinterne Ereignisse in eine Vorgänger-/Nachfolger-Beziehung gebracht. Transaktionen beschreiben die Verknüpfung von Aufgaben zwischen zwei betrieblichen Objekten (siehe vorherigen Teilabschnitt).

Die Bedeutung der Bausteine *Aufgabe* und *internes Ereignis* spiegelt sich in den daraus entwickelten Komponenten der RESTful SOA wider:

- *Elementare Vorgangsressourcen* repräsentieren die Funktionalität zur Durchführung betrieblicher Aufgaben. Der *Workflow* einer Vorgangsressource steuert das Zusammenwirken von Entitätsressourcen durch Methodenaufrufe und spezifiziert dadurch das Lösungsverfahren der Aufgabe (Realisierung aus Innensicht durch einen VOT).
- *Nicht-elementare Vorgangsressourcen* stellen die Logik zur Durchführung eines Teils oder des gesamten Vorgangsnetzes der betrieblichen Objekte bereit. Die VOTs spezifizieren hierfür Workflows, welche eine Menge von Ressourcen orchestrieren. Die Gestaltung einer Choreographie zwischen Vorgangsdiensten fußt auf den Transaktionsbeziehungen im Geschäftsprozessmodell.
- *Interaktionsbeziehungen* beschreiben die Ausführungsreihenfolge von Vorgangsressourcen gemäß den objektinternen Ereignissen zwischen Aufgaben betrieblicher Objekte. Die Ablaufbeziehungen der Vorgänge (Geschäftsprozess) bilden den Rahmen für die Gestaltung von Choreographie oder Orchestrierung der Vorgangsressourcen.

Beispiel: Die vorausgegangenen Ausführungen werden anhand der betrieblichen Aufgaben *>Flugwahl* und *Angebot>* des betrieblichen Objekts *Vertrieb* erläutert (Abbildung 4.11). Beide Aufgaben werden jeweils in einen VOT des VOS *Vertrieb* überführt, ihre Eigenschaften spezifiziert und ihnen IOTs für die Kommunikation mit dem Kunden zugeordnet (Abbildung 5.13). Das objektinterne Ereignis zwischen *>Flugwahl* und *Angebot>* definiert die Nachricht der interacts_with-Beziehung, mit der die Durchführung von VOT *Angebot>* ausgelöst wird. Die Funktionalität eines Vorgangs wird in der RESTful SOA in Form von zwei Vorgangsdiensten (bzw. ihren Individual- und Listenressourcen) veröffentlicht. Im Zuge der Konsolidierung werden diese beiden Dienste anhand des MEP *In-Out* zusammengefasst und mit der abgeleiteten Ressourcenspezifikation abgestimmt (Abbildung 5.16). Die Vorgangsressourcen der Angebotsabwicklung realisieren somit die Durchführung der unterstützten GP-Aufgaben (*>Flugwahl*, *Angebot>*).

Betriebliche Leistung und betriebliches Objekt

Die Zusammenhänge der modellbasierten Überführung der GP-Bausteine *betriebliche Leistung* und *betriebliches Objekt* in die Komponenten der RESTful SOA werden in Abbildung 5.25 dargestellt. Die Überarbeitung der Struktursicht erfolgt konform zu den Konsolidierungsschritten der betrieblichen Transaktionen.

Sowohl die **betriebliche Leistung** als auch das **betriebliche Objekt** werden in LOTs bzw. OOTs des KOS überführt und konsolidiert. Anschließend werden die konzeptuellen Objekttypen in Entitätsressourcen und Dokumenttypen der initialen Spezifikation der RESTful SOA transformiert (Außensicht). Die Realisierung der Entitätsressourcen spezifizieren EOTs, die zur Persistierung ihrer Zustände auf DOTs zugreifen (Innensicht).

Neben der betrieblichen Leistung stellen betriebliche Objekte häufig eine fachliche Quelle zur Identifikation von Stammdaten des Anwendungssystems dar. Zudem spielt das betriebliche Objekt in den verhaltensorientierten Sichten des Modellsystems eine wichtige Rolle. So stellt ein betriebliches Objekt den Ausgangspunkt für die Ableitung von VOS bzw. OS-VD im Anwendungsmodell sowie REST-Architekturmodell dar. Das OS-VD skizziert den Rahmen, in dem die Abgrenzung nicht-elementarer Vorgangsdienste und die detaillierte Gestaltung der Service-Koordination erfolgen.

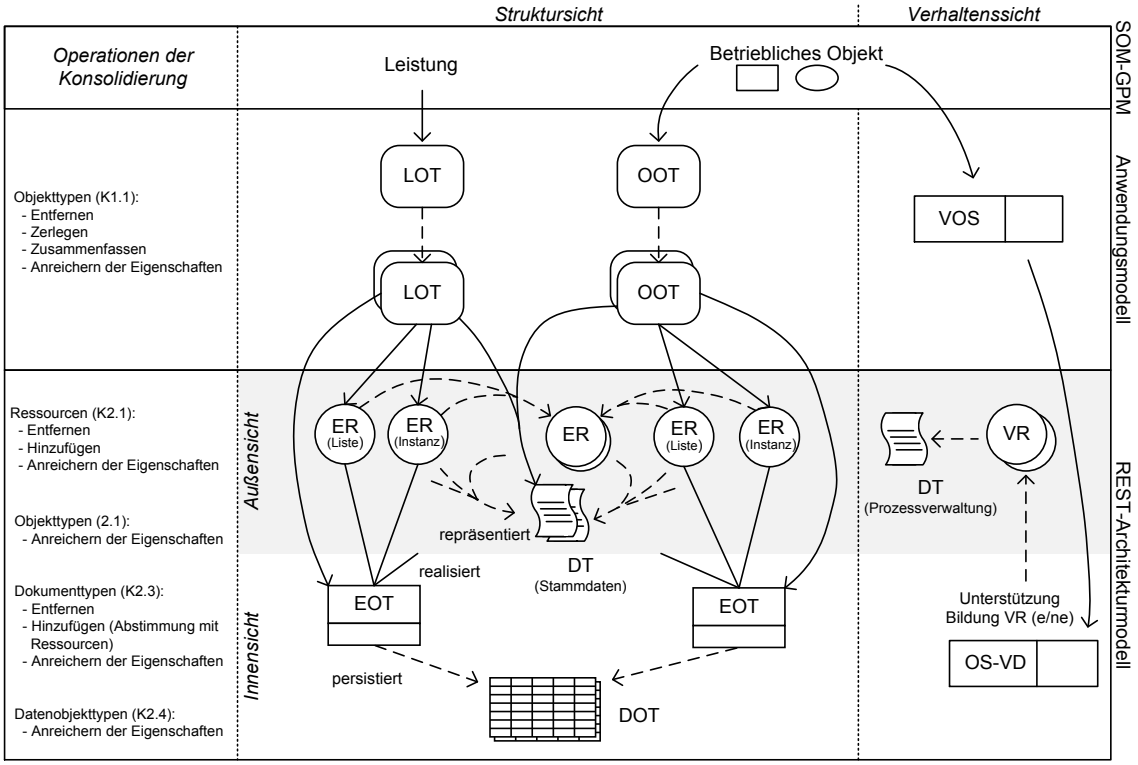


Abbildung 5.25: Beziehungen zwischen Leistung und betrieblichem Objekt sowie der RESTful SOA

Die charakteristischen Merkmale von betrieblichem Objekt und betrieblicher Leistung sowie ihre Entsprechung in der Spezifikation der RESTful SOA werden nachfolgend untersucht:

- Geschäftsprozesse erstellen betriebliche Leistungen und übergeben diese an den beauftragenden „Kunden“ [FeSi13, S. 200].
- Betriebliche Objekte kapseln eine Menge von Aufgaben, die zusammengehörige Ziele verfolgen und auf einem gemeinsamen Aufgabenobjekt arbeiten [FeSi13, S. 203].

Die Bedeutung von Leistung und betrieblichen Objekten des SOM-Geschäftsprozessmodells spiegelt sich in den auf dieser Basis entwickelten Komponenten der RESTful SOA wider:

- *Entitätsressourcen* realisieren Dienste zur Verwaltung von Stammdaten, welche die Erstellung und Übergabe der Leistung des Geschäftsprozesses sowie die Merkmale der beteiligten betrieblichen Objekte beschreiben.
- *Stammdatenspezifische Dokumenttypen* definieren die Struktur der übertragenen Informationen von Leistung und betrieblichen Objekten. Sie repräsentieren die Zustände von Entitätsressourcen und werden über Nachrichten zwischen Vorgangsdiensten zur Durchführung eines konkreten Geschäftsvorfalls ausgetauscht.
- Das *OS-VD* umfasst eine Menge von (ne/e) VOTs und veröffentlichten Ressourcen, die zusammengehörige Sach- und Formalziele eines betrieblichen Objekts verfolgen und dazu auf einer gemeinsamen Menge von Entitätsressourcen (objektinterner Speicher) arbeiten. Im Prozess der Spezifikation von RESTful SOA spielt es primär eine unterstützende Rolle.

Beispiel: Die Leistung *Flugticketverkauf* sowie das betriebliche Objekt *Kunde* des Geschäftsprozesses der Fallstudie werden in einen LOT bzw. OOT des KOS überführt und konsolidiert (Abbildung 5.13). In der RESTful SOA kapseln jeweils zwei Entitätsressourcen (Listen- und Individualressource) die Geschäftslogik zur Verwaltung von Details der Leistungsspezifikation und des Kunden (Abbildung 5.15). Aus der Innensicht wird die Schnittstelle und Funktionalität der Ressourcen durch zugehörige EOTs implementiert. Die Struktur der persistenten Datenhaltung definieren die DOTs *Kunde* und *Leistungsspezifikation* (Abbildung 5.17). Zur Repräsentation der Stammdaten werden Dokumenttypen abgeleitet (Abbildung 5.14). Das Diskursweltobjekt *Flugbetrieb* wird in ein VOS des Anwendungsmodells und anschließend ein OS-VD des REST-Architekturmodells überführt (Abbildung 5.13 und Abbildung 5.16). Hinweise zur Abgrenzung von Vorgangsdiensten geben die Interaktionsbeziehungen zu anderen OS-VDs. So wird der Dienst *Flugauskunft* anhand der Beziehungen zum OS-VD *Vertrieb* abgegrenzt.

Kernaussagen zur modellbasierten Entwicklung mit der SOM-R-Methodik

Auf Basis der dargelegten konzeptuellen Erkenntnisse werden fünf Kernaussagen über die modellbasierte Entwicklung von RESTful SOA aus SOM-Geschäftsprozessmodellen formuliert:

- Die RESTful SOA veröffentlicht eine Menge von Ressourcen, die sich aus den betrieblichen Transaktionen, Aufgaben, Leistungen und betrieblichen Objekten in SOM-GPMs ableiten. Der fachliche Inhalt und die Lösungsverfahren der spezifizierten RESTful Services bzw. ihrer Ressourcen werden auf der Aufgabenebene begründet.
- Die Entitätsressourcen sind weitestgehend auf Grundlage der verwendeten Begrifflichkeiten und Semantik von betrieblichen Transaktionen, Leistungen und betrieblichen Objekten zugreif- und nutzbar. Die einheitliche Ressourcenschnittstelle definiert das Lesen, Einfügen, Aktualisieren oder Löschen von Transaktions- und Stammdaten, und folglich von den spezifischen Ausführungszuständen im Geschäftsablauf.
- Vorgangsressourcen beschreiben Workflows, deren Ausführung über die einheitliche Schnittstelle verwaltet werden kann. Die Zugriffssemantik auf die Vorgangsressourcen einer RESTful SOA ist dadurch fest vorgegeben. Die zielgerichtete Nutzung von Ressourcen erfordert damit insbesondere die Kenntnis ihres fachlichen Inhalts.

- Die Koordination lose gekoppelter Ressourcen basiert auf dem Austausch von Dokumenten, die Ausführungszustände des Geschäftsvorfalls repräsentieren. Den Ausgangspunkt für die nachrichtenbasierte Interaktion und ihre fachliche Bedeutung bilden die betrieblichen Transaktionen des SOM-Geschäftsprozessmodells.
- Die Spezifikation von Innen- und Außensicht der RESTful SOA wird im REST-Architekturmodell weiterhin nach fachlichen Abhängigkeiten der Geschäftsprozessebene strukturiert. Nach dem Entwicklungsvorgehen der SOM-R-Methodik wird die Realisierung von Ressourcen anhand der fachlichen Kriterien von den Objekttypen der REST-Architektur gekapselt (z. B. EOT und Ressourcen zur Kundenverwaltung). Technische Aspekte wie die Realisierung von Diensten in Form verteilter Komponenten des AwS, und damit ein Aufweichen der fachlichen Abhängigkeiten, werden im Implementierungsmodell adressiert.

Die vorausgegangenen Ausführungen zeigen, dass die SOM-R-Methodik als Lösungsansatz geeignet ist, das erste Untersuchungsziel der modellbasierten Gestaltung von RESTful SOA aus SOM-Geschäftsprozessmodellen zu erfüllen. Durch die Anwendung der SOM-R-Methodik wird eine Überführung von Semantik und Begrifflichkeiten der SOM-Geschäftsprozessebene auf die softwaretechnische REST-Ebene gefördert.

5.6.2 Prüfung der Formalziele

Der Konstruktion eines Lösungsansatzes werden in der vorliegenden Arbeit drei Formalziele an die Seite gestellt (vgl. Abschnitt 1.2.2). Die SOM-R-Methodik des fünften Kapitels stellt den Lösungsansatz für das erste Untersuchungsziel der Arbeit dar. Das Erreichen der definierten Formalziele wird im Folgenden geprüft:

- Die SOM-R-Methodik fördert die *Bewältigung der Komplexität* der Systementwicklungsaufgabe dieser Arbeit sowie des im Zuge einer Aufgabendurchführung zu spezifizierenden Modellsystems durch die Bereitstellung von: (1) SOM-R-Architekturmodell, das durch Bildung von Sichten und hierarchisch angeordneten Modellebenen das Modellsystem in Teilsysteme untergliedert und damit die Komplexitätsbewältigung unterstützt (Abschnitt 5.2). Zudem hilft die Bereitstellung von Modellierungsansätzen bei der Beherrschung von subjektiven Einflüssen auf die Modellbildung (Abschnitt 5.4). Die Durchführung des Entwicklungsprozesses unterstützen das (2) SOM-R-Vorgehensmodell (Abschnitt 5.3) sowie die (3) Definition von Transformationen und Konsolidierungen als Schritte der systematischen Modellerstellung (Abschnitt 5.5). Der Systementwickler bzw. Modellierer wird bei der Durchführung der nicht-automatisierbaren Entwicklungsaufgaben durch die Bereitstellung von Konsolidierungsschritten methodisch angeleitet und damit eine erfolgreiche Bearbeitung der bestehenden Systementwicklungs-/Modellierungsaufgabe weiter unterstützt.
- Die *konsistente Abstimmung* innerhalb und zwischen den verschiedenen Abstraktionsebenen des zu spezifizierenden Modellsystems (Systemspezifikation), wird in der SOM-R-Methodik durch die Bereitstellung von Hilfsmitteln gefördert: (1) Auf jeder Ebene des SOM-R-Architekturmodells definiert ein integriertes Metamodell die verfügbaren Metaobjekte und Beziehungen. Die Ebenen des SOM-R-Modellsystems

sind damit auf Konsistenz und Vollständigkeit bezüglich des eingesetzten Metamodells überprüfbar (Abschnitt 5.2). (2) Die erlaubten Abbildungsbeziehungen zwischen den Metaobjekten unterschiedlicher Modellebenen werden in Beziehungsmetamodellen definiert. Diese beschreiben zusammen mit den Transformationsspezifikationen (T1-T3) die automatisierte Ableitung von konsistenten Zielmodellen aus den Quellmodellen benachbarter Modellebenen (Abschnitt 5.5). (3) Bei der Durchführung von nicht-automatisierten Entwicklungsschritten wird die Erzeugung konsistenter Modelle durch die Definition von Konsolidierungsoperationen für die jeweilige Modellebene gefördert (Abschnitt 5.5). Der konzipierte REST-Modellierungsansatz ist auf das Begriffssystem sowie die übergeordnete Metapher der SOM-R-Methodik abgestimmt. Auf dieser Basis kann sichergestellt werden, dass die modellierten Bausteine auf den Ebenen des Modellsystems stets von Bausteinen der höheren Abstraktionsebenen existenzabhängig sind (Abschnitt 5.5).

Damit die Ausrichtung von Komponenten der RESTful SOA auf das SOM-Geschäftsprozessmodell überprüft werden kann, muss zusätzlich die Nachvollziehbarkeit der Modellbildung gewährleistet werden. Hierfür stellt die Methodik bisher nur einen Architekturrahmen bereit. Ein Ansatz zur Dokumentation von Entwicklungsschritten in der SOM-R-Methodik wird in Abschnitt 6.3 vorgeschlagen.

- Das Ziel der *Flexibilität* oder auch der flexiblen Anpassbarkeit der Entwicklungsmethodik wird durch den modularen Aufbau von Architektur- und Vorgehensmodell erreicht. Der konsequent hierarchische Aufbau des Modellsystems erlaubt grundsätzlich das Anpassen (Hinzufügen, Entfernen oder Ändern) von Modellebenen. Des Weiteren stützt die Trennung von automatisierten und nicht-automatisierten Entwicklungsschritten die Adaption der Transformationen bzw. Konsolidierungen an veränderte Anforderungen. Durch die Trennung von REST-Architekturmodell und Implementierungsmodell wird zudem die *Plattformunabhängigkeit* der spezifizierten RESTful SOA gefördert. Das REST-Architekturmodell ist hier generell unabhängig von der Wahl einer konkreten Basismaschine. Dem Konzept der modellgetriebenen Entwicklung folgend, ist die automatisierte Erzeugung von Programmcode der RESTful SOA Gegenstand des dritten Transformationsschrittes der Methodik. Er kapselt die plattformspezifischen Details der Systemspezifikation (Abschnitte 5.2 und 5.5.6).

Abschließend ist festzuhalten, dass der Modellierer bei der Erfassung von Struktur und Verhalten eines Systems, der Herstellung von Konsistenz und Vollständigkeit sowie von Struktur- und Verhaltenstreue im zu erstellenden Modellsystem, und der Sicherstellung einer zweckorientierten Modellierung unterstützt wird (Abschnitt 1.2.2). In Bezug auf das erste Untersuchungsziel erfüllt die SOM-R-Methodik somit die aufgestellten Formalziele.

5.7 Zusammenfassung

Gegenstand des fünften Kapitels ist die Konstruktion einer Entwicklungsmethodik zur Unterstützung der Systementwicklung bei der modellbasierten Gestaltung von RESTful SOA auf Basis von SOM-Geschäftsprozessmodellen (Teilziel 1). Der verfügbare Lösungsraum für die Durchführung der Konstruktion wird in der vorliegenden Arbeit insbesondere durch zwei

Restriktionen begrenzt. So bildet die SOM-Methodik das *methodische Fundament* des Lösungsansatzes. Des Weiteren ist die Spezifikation von RESTful SOA als *Zielarchitektur* der modellbasierten Systementwicklung gegeben.

Die SOM-R-Methodik bildet das Ergebnis der Konstruktion. Im Vergleich zu modellgetriebenen Ansätzen verwandter Arbeiten, deren Fokus häufig auf den plattformnahen Abstraktionsebenen und der Ablaufsicht von Geschäftsprozessmodellen liegt, nimmt die SOM-R-Methodik eine ganzheitliche Sichtweise auf die Entwicklung von Informationssystemen ein. Dem Systementwickler gibt die Methodik ein Architekturmodell, ein Vorgehensmodell, einen Modellierungsansatz für RESTful SOA (Metapher und Metamodell) sowie definierte Entwicklungsschritte als Hilfsmittel zur Modellbildung an die Hand. Der definierte Entwicklungsprozess sieht die schrittweise Gestaltung der Ebenen des Modellsystems als Folge von Schema-Transformationen (T1-T3) und Schema-Konsolidierungen (K1-K3) vor. Transformationen kapseln das automatisierbare Entwicklungswissen zur schrittweisen Verringerung des Abstraktionsgrades im Modellsystem und beschreiben die Erzeugung der initialen Schemata einer Modellebene. Die Generierung des plattformspezifischen Programmcodes der RESTful SOA erfolgt im Zuge der dritten und letzten Transformation. Die Überarbeitung der Modellebenen der initial abgeleiteten fachlichen AwS-Spezifikation und des REST-Architekturmodells ist Aufgabe der Entwickler. Die definierten Konsolidierungen unterstützen die beteiligten Personen bei der Durchführung der nicht-automatisierbaren Entwurfsschritte und Entscheidungen und geben hierfür einen methodischen Rahmen sowie Handlungsanweisungen vor. Die durchgängige modellbasierte Spezifikation des IS führt zu einer Anhebung der Abstraktionsebene im Systementwicklungsprozess.

Ein wichtiges Hilfsmittel, um die Systementwicklung in einem durchgängig modellbasierten Prozess zu realisieren, ist die fachliche AwS-Spezifikation als „Zwischenebene“ im SOM-R-Architekturmodell. Sie unterstützt die Abbildung von Konzepten der Aufgabenebene auf die softwaretechnische Aufgabenträgerebene. Als Beispiel sei die Überführung betrieblicher Transaktionen zunächst in TOTs sowie in Interaktionsbeziehungen zwischen VOTs der (objektorientierten) fachlichen AwS-Spezifikation genannt. Diese werden in Entitätsdienste sowie in Dokumenttypen und Datenobjekttypen der RESTful SOA abgebildet.

Eine konsistente Abstimmung der Modellebenen im Gesamtmodellsystem ermöglicht die Anwendung des konzipierten Entwicklungsvorgehens der Methodik. Durch die explizite Definition struktureller Überarbeitungsschritte in den Konsolidierungen wird sichergestellt, dass die Existenz aller Bausteine der RESTful SOA von Elementen des SOM-GPM abhängig ist. Die Anwendbarkeit der SOM-R-Methodik wird anhand der eingeführten Fallstudie demonstriert. Die konkrete Ausgestaltung und Spezifikation der Implementierungsebene wird am Beispiel zweier Basismaschinen gezeigt.

6 Modellbasierte Pflege von Informationssystemen mit der SOM-R-Methodik

Gegenstand des sechsten Kapitels ist die Konstruktion eines Lösungsansatzes für die methodisch geleitete Anpassung der Spezifikation von RESTful SOA an veränderte Anforderungen eines strukturflexiblen SOM-Geschäftsprozessmodells. Die SOM-R-Entwicklungsmethodik wird hierzu um Hilfsmittel erweitert, welche die Durchführung der Systementwicklungsaufgabe der modellbasierten Pflege betrieblicher Informationssysteme unterstützen. Mit der erweiterten Methodik wird das zweite Untersuchungsziel erreicht. Sie stellt den Lösungsansatz für das Konstruktionsproblem der vorliegenden Arbeit dar.

Die Ausführungen des Kapitels beginnen mit der Erläuterung von Rahmenbedingungen und Anforderungen an die Aufgabe der Weiterentwicklung von Systemen sowie der Bestimmung von Bestandteilen, dem Aufbau und Vorgehen der durchzuführenden Methodikerweiterung. Auf dieser Grundlage erfolgt schrittweise die Konzeption von Dokumentations-, Modellvergleichs- und Empfehlungsansatz sowie eines Vorgehensmodells als Bestandteile eines integrierten Lösungsansatzes zur modellbasierten Systempflege. Eine Zusammenfassung und Diskussion der erzielten Ergebnisse schließt das Kapitel ab.

6.1 Anforderungen an die Erweiterung der SOM-R-Methodik

Die SOM-R-Methodik des fünften Kapitels bildet die methodische Basis für die Konstruktion eines Lösungsansatzes zur modellbasierten Pflege von Informationssystemen. In Abschnitt 6.1 werden zunächst die methodischen Rahmenbedingungen der Problemstellung zusammengefasst (Abschnitt 6.1.1), die allgemeine Aufgabe der Weiterentwicklung von Systemen anhand eines Modells strukturiert (Abschnitt 6.1.2) und anschließend die Anforderungen an die Erstellung des Lösungsansatzes dieser Arbeit zur Systempflege bestimmt (Abschnitt 6.1.3).

6.1.1 Methodische Rahmenbedingungen

Die methodischen Rahmenbedingungen des zu lösenden Konstruktionsproblems werden insbesondere durch die Bestandteile und Merkmale der SOM-R-Entwicklungsmethodik sowie Strukturflexibilität als Auslöser der Weiterentwicklung von existierenden Systemen bestimmt.

STRUKTURFLEXIBILITÄT AUF DER SCHEMAEBENE

Flexible SOM-Geschäftsprozessmodelle sind der Ausgangspunkt für die Weiterentwicklung eines spezifizierten SOM-R-Modellsystems (Abschnitt 4.2.3). In der vorliegenden Arbeit wird ausschließlich die Bewältigung von geänderten Anforderungen in *strukturflexiblen* SOM-Geschäftsprozessmodellen untersucht (Abschnitt 1.2.1). Strukturflexibilität manifestiert sich im Geschäftsprozessmodell als Folge einer Anwendung der elementaren Änderungsoperationen *Hinzufügen* und/oder *Entfernen* von Modellelementen, um dieses evolutionär anzupassen. Die zu behandelnden Veränderungen sind zudem auf die *Schemaebene* des Modellsystems

beschränkt (Abschnitt 6.4.1)**.** Modelländerungen, die das Resultat von Anpassungen auf Metaebene oder Meta-Metaebene darstellen, werden nicht weiter untersucht.

MODELLBASIERTE SYSTEMENTWICKLUNG

Die Unterstützung einer *vollständig modellbasierten Durchführung der Systementwicklungsaufgabe* dieser Arbeit ist ein zentrales Merkmal der SOM-R-Methodik aus Kapitel 5. Für die hierbei bestehende Aufgabe der Modellbildung werden jeweils ein integriertes Metamodell und Metapher sowie eine Menge von automatisierbaren (Transformation) und nicht-automatisierbaren (Konsolidierung) Entwicklungsschritten auf den Ebenen des SOM-R-Architekturmodells bereitgestellt (Abschnitte 5.4 und 5.5).

TOP-DOWN-VORGEHEN

Das SOM-R-Vorgehensmodell beschreibt die *Reduzierung des Abstraktionsgrades* der Systemspezifikation als schrittweisen Ablauf des Entwicklungsprozesses (Abschnitt 5.4). Die Implementierung des REST-Architekturmodells wird dabei *top-down* aus der Ebene des SOM-Geschäftsprozessmodells erzeugt. Den Aufbau des Modellsystems bzw. die Anordnung der zu spezifizierenden Modellebenen legt die Modellebenenhierarchie des SOM-R-Architekturmodells fest. Da Strukturflexibilität in SOM-Geschäftsprozessmodellen (höchste Abstraktionsebene) der Auslöser einer Systempflege in dieser Arbeit ist, bietet sich für die Anpassung des Modellsystems ebenfalls die Wahl eines *Top-Down-Vorgehens* im zu konzipierenden Lösungsansatz an.

ZUSAMMENSPIEL METHODISCHER KONZEPTE VON SOM UND RESTFUL SOA

Die zentralen methodischen Konzepte für die Modellierung von SOM-Geschäftsprozessmodellen sind das *betriebliche Objekt* und die *transaktionsorientierte Koordination von Aufgaben lose gekoppelter Objekte* (Abschnitt 4.1.2). Auf der Ebene des REST-Architekturmodells stellen die *einheitliche REST-Schnittstelle* und *Ressourcenorientierung* die Grundlage für die Gestaltung der Zielarchitektur dar. Die SOM-R-Methodik spezifiziert das Lösungsverfahren zur modellbasierten Überführung von Konzepten der Aufgabenebene (SOM-GPM) in die Aufgabenträgerebene (REST-Architekturmodell). Die Erstellung von Ressourcen mit einheitlicher Schnittstelle und ihre Nutzung basieren auf Semantik und Begrifflichkeit eines oder mehrerer Schemaobjekte des SOM-Geschäftsprozessmodells (Abschnitt 5.6).

EXISTENZABHÄNGIGKEITEN ZWISCHEN SCHEMABAUSTEINEN

Die Bestandteile der Aufgabenträgerebenen des SOM-R-Architekturmodells stehen in *existenziellen Abhängigkeitsbeziehungen* zu den fachlichen Anforderungen der Aufgabenebene. Der Grund für diese Abhängigkeit liegt in der Gestaltung der strukturellen Überarbeitungsschritte in den Konsolidierungen. So ist definiert, dass die Bausteine einer Modellebene nur aus bereits existierenden Bausteinen eines zu konsolidierenden (initialen) Schemas erstellt werden können. Da die initialen Schemata wiederum automatisiert in einer Transformation erzeugt werden, können folglich die Elemente einer Abstraktionsebene (Zielebene) zu den Elementen der nächst-höheren Ebene (Quellebene) in Beziehung gesetzt werden. Jeder Baustein des SOM-Geschäftsprozessmodells spiegelt sich somit in Komponenten der RESTful SOA wider

(Abschnitt 5.6). Die existenziellen Abhängigkeitsbeziehungen im SOM-R-Modellsystem können als Basis für die Analyse der Auswirkungen von Strukturänderungen dienen.

6.1.2 Aufgabenmodell der Weiterentwicklung von Systemen

Die Aufgabe der Weiterentwicklung von existierenden (entwickelt und eingeführten) Informationssystemen wird nachfolgend in Form einer Spezialisierung des allgemeinen Aufgabenmodells der Systementwicklung[54] strukturiert [FeSi13, S. 497 f.]. Im Kontext der *Weiterentwicklung* von Informationssystemen (Systemweiterentwicklung) wird in dieser Arbeit auch der Begriff der *Systempflege* verwendet (Abschnitte 2.2.1 und 2.2.3).

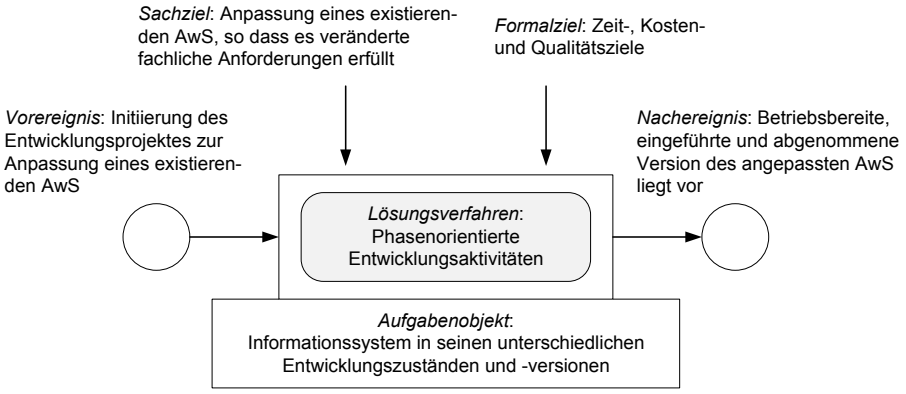


Abbildung 6.1: Aufgabenmodell der Systemweiterentwicklung (in Anlehnung an [FeSi13, S. 497], eigene Anpassungen)

Das Aufgabenmodell der Systemweiterentwicklung zeigt Abbildung 6.1. Die Bestandteile des Aufgabenmodells werden im Folgenden erläutert ([FeSi13, S. 497 f.], angepasste Erläuterung):

- *Sachziel* der Systemweiterentwicklungsaufgabe ist die Anpassung eines implementierten und eingeführten AwS an veränderte Anforderungen.
  Das AwS ist in der vorliegenden Arbeit als RESTful SOA realisiert.
- *Formalziele* beschreiben Zeit-, Kosten- und Qualitätsziele des durchzuführenden Entwicklungsprozesses.
- Das *Vorereignis* löst die Durchführung der System(weiter)entwicklungsaufgabe aus. Ausgehend von Veränderungen in den Systemanforderungen wird ein Entwicklungsprojekt zur Anpassung der bestehenden Version des AwS initiiert.
  Änderungen in *strukturflexiblen* SOM-GPMs sind in dieser Arbeit das auslösende Ereignis zur Initiierung von Entwicklungsprojekten.
- Als *Nachereignis* liegt das angepasste AwS in einer neuen Version vor. Dieses ist konsistent mit den Beschreibungsebenen des Informationssystems abgestimmt.

[54] Unter dem Begriff *Systementwicklung* werden in der vorliegenden Arbeit sowohl die Neu-Entwicklung von AwS auf den Beschreibungsebenen des Informationssystems, als auch die Weiterentwicklung existierender AwS subsumiert. Die Aufgabe der Systemweiterentwicklung wird hierbei als Spezialisierung der allgemeinen Systementwicklungsaufgabe im Hinblick auf die Anpassung des gesamten Informationssystems verstanden. Das Lösungsverfahren zur Durchführung der Systementwicklungsaufgabe beschreibt in dieser Arbeit die SOM-R-Methodik (Kapitel 5).

- Das Informationssystem (oder Teil des IS) in seinen unterschiedlichen Entwicklungszuständen und Versionen bildet das *Aufgabenobjekt*. Die Entwicklungszustände beschreiben die Systemspezifikation auf allen Abstraktionsebenen sowie die relevante Systemumgebung.
  Das Aufgabenobjekt setzt sich in dieser Arbeit aus SOM-Geschäftsprozessmodell, fachlicher AwS-Spezifikation, softwaretechnischem REST-Architekturmodell und der Implementierung der RESTful SOA in Form von Software- und Hardwarekomponenten zusammen. Zusätzlich sind die Vor-Versionen der AwS-Entwicklungszustände und die zu erstellende neue Systemversion als Teil des Aufgabenobjekts zu erfassen.
- Das *Lösungsverfahren* der Systemweiterentwicklungsaufgabe wird als zielgerichtete Anpassung des AwS in Form einer phasenorientierten Durchführung von Entwicklungsaktivitäten spezifiziert.

LÖSUNGSVERFAHREN DER SYSTEMWEITERENTWICKLUNGSAUFGABE

Gemäß den drei Hauptphasen zur Durchführung der Systementwicklung sieht das Lösungsverfahren der Systemweiterentwicklung ebenfalls eine phasenorientierte und damit schrittweise Anpassung der IS-Beschreibungsebenen an geänderte Anforderungen vor (Abschnitt 2.2.3). Jede Phase lässt sich wiederum in die beiden Hauptaktivitäten *Analyse und Durchführung der System-Pflege* untergliedern (nach [FeSi13, S. 498 f.], angepasste Erläuterung):

- *Anforderungsanalyse und -definition (A)*: Veränderungen im Modell des betrieblichen Objektsystems sind der Ausgangspunkt für die Identifikation neuer (hinzugefügte) oder veralteter (entfernte) Anforderungen auf der Aufgabenebene des Informationssystems. Auf Basis der geänderten Anforderungen erfolgen die Analyse und Durchführung einer zielgerichteten Anpassung der Anwendungsfunktionen im Anwendungsmodell, um diese konsistent mit dem betrieblichen Objektsystem abzustimmen.
- *Softwareentwurf (D)*: Ausgehend von dem veränderten Anwendungsmodell wird die bestehende Spezifikation der Softwarearchitektur analysiert und weiterentwickelt. Das Ergebnis der Phase ist die Softwarearchitektur, deren Komponenten konsistent auf die veränderten Anforderungen der fachlichen Ebene ausgerichtet sind.
- *Realisierung (R)*: In der letzten Phase wird der Programmcode des AwS an die Spezifikation der veränderten Softwarearchitektur angepasst.

6.1.3 Bestandteile der erweiterten SOM-R-Methodik

Das *SOM-R-Architekturmodell*, die *Modellierungsansätze* der Modellebenen sowie die *Transformations-* und *Konsolidierungsschritte* bilden die methodische Basis für die Erweiterung der SOM-R-Methodik (Abschnitte 5.2 bis 5.5). Gegenstand der Untersuchung ist die Pflege von existierenden Modellsystemen (Schemaebene).

Die Pflege eines spezifizierten Modellsystems erfolgt von-oben-nach-unten entlang der Beschreibungsebenen des Informationssystems. Das Ergebnis der Pflege ist das weiterentwickelte Anwendungssystem RESTful SOA, das konsistent auf die Geschäftsprozessebene sowie die weiteren Abstraktionsebenen des gesamten IS abgestimmt ist. Der grundlegende

Ablauf des Lösungsverfahrens der Systemweiterentwicklung umfasst die Kernaktivitäten A, D und R und wird in Abbildung 6.2 in vereinfachter Form dargestellt. Die Beschreibungsebenen sind auf die Ebenen des SOM-R-Architekturmodells ausgerichtet.

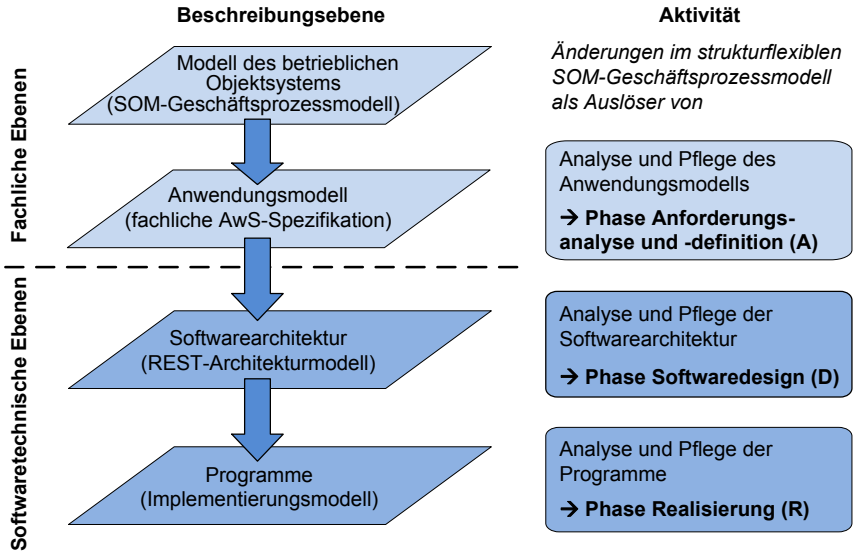


Abbildung 6.2: Kernaktivitäten des Lösungsverfahrens der Systemweiterentwicklung (in Anlehnung an [FeSi13, S. 482 f.], eigene Anpassung)

Die Analyse, Gestaltung und Pflege sowie die Dokumentation von Informationssystemen sind als zentrale Aufgaben im Rahmen der Systementwicklung durchzuführen (vgl. Abschnitt 1.1). Entsprechend dieser Aufgabenstrukturierung wird für den zu erstellenden Lösungsansatz dieser Arbeit ebenfalls die Bereitstellung von Hilfsmitteln zur Unterstützung der Aktivitäten *Analyse und Durchführung der Pflege* des SOM-R-Modellsystems sowie einer begleitenden *Dokumentation* des Entwicklungsvorgehens angestrebt.

Das Ziel der **Analyse** ist die Identifikation von Unterschieden sowie die Ermittlung von Anforderungen, um in der darauffolgenden Pflege des Modellsystems die Anpassung seiner Bausteine (Komponenten des AwS) zielgerichtet vornehmen zu können. Die Dokumentation des Modellsystems in seinen verschiedenen Entwicklungszuständen unterstützt die Durchführung der Analyse- und Pflegeaktivitäten.

Die **Pflege** des SOM-R-Modellsystems wird, entsprechend den Entwicklungsschritten zur Modellbildung in der SOM-R-Methodik, durch das wiederholte Durchlaufen der Transformationsschritte (T1-3) und Konsolidierungsschritte (K1-3) realisiert. Das grundlegende Vorgehen veranschaulicht Abbildung 6.3. *Transformationen* ermöglichen die Bewältigung von Strukturflexibilität durch die automatisierte (a) „Neu-Transformation“ der veränderten Quell- in angepasste Zielschemata[55]. Die *Konsolidierung* der initialen Schemata einer Modellebene erfordert, wie auch bei der Neu-Entwicklung von Systemen, bei der Anpassung bestehender

[55] Die Transformation der konsolidierten Schemata einer Modellebene (E_N) in die initialen Schemata der nächst-tieferen Modellebene (E_{N+1}) erfolgt somit durch Neu-Generierung. Dieser Ansatz wird auch in der modellgetriebenen Entwicklung zur Bewältigung von Veränderungen in Quellmodellen angewendet (z. B. [SVE+07, S. 13], [PeMe06, S. 135]).

Systeme das Treffen von nicht automatisierbaren Entwurfsentscheidungen (na) durch den Systementwickler.

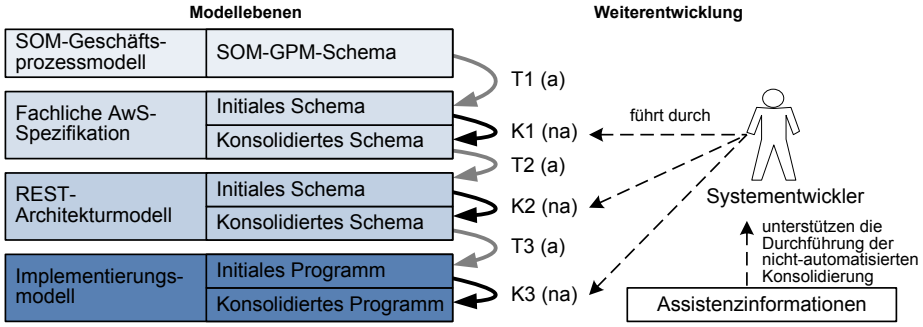


Abbildung 6.3: Entwicklungsschritte zur Pflege von SOM-R-Modellsystemen

Zentrales Ziel bei der Konzeption des Lösungsansatzes ist die Bereitstellung von Hilfsmitteln, um den Systementwickler bei der Durchführung der nicht-automatisierbaren Aktivitäten der Systementwicklungsaufgabe und die Bewältigung der damit einhergehenden Komplexität zu unterstützen. Auf Basis der Hilfsmittel sollen Informationen erzeugt werden, die dem Systementwickler bei der Schema-Konsolidierung im Kontext der Systempflege assistieren. Sie werden deshalb im Folgenden als **Assistenzinformationen** bezeichnet.

Ein wichtiges Hilfsmittel ist dabei die **Dokumentation** des Entwicklungswissens und der Entwicklungsartefakte. Neben den Schemata des spezifizierten Modellsystems entstehen auch im Rahmen Durchführung von Transformation und Konsolidierung zentrale Entwicklungsinformationen. Da die Erzeugung von initialen Schemata im SOM-R-Modellsystem automatisiert erfolgt, sind besonders die Dokumentation von konsolidierten Schemata und der personell getroffenen (na) Entwurfsentscheidungen der Konsolidierung (K1-3) von hervorgehobenem Interesse für die Systementwicklung (Abbildung 6.3). Die *durchgängige Dokumentation* des Entwicklungsprozesses ist somit eine wichtige Aufgabe in der modellbasierten Systemweiterentwicklung.

Eine Voraussetzung für die Durchführung der zielgerichteten Anpassung eines Modellsystems durch einen Entwickler ist die Identifikation von hinzugefügten oder entfernten Bausteinen im strukturflexiblen SOM-GPM sowie auf den Beschreibungsebenen der fachlichen AwS-Spezifikation, dem REST-Architekturmodell und Implementierungsmodell. Die *Ermittlung des Modell-Deltas* auf diesen Modellebenen erfordert die Bereitstellung eines methodischen Hilfsmittels, welches die Durchführung von **Modellvergleichen** unterstützt.

Bei der Durchführung der *Pflege von Schemata* auf den Ebenen des SOM-R-Architekturmodells sind Entscheidungen zu treffen, wie veränderte initiale Schemata in eine neue Version der korrespondierenden konsolidierten Schemata zu überführen sind. Die Analyse von Modelldifferenzen und die Bestimmung der Auswirkungen von Modellanpassungen stellen den Systementwickler vor große Herausforderungen. In der SOM-R-Methodik sind auf Basis der Metamodelle für jede Modellebene die potenziellen Typen von Modelländerungen bestimmbar. Anhand der Metamodelle und dem Modell-Delta können somit für möglichen Änderungstypen einer Modellebene Empfehlungen aufgestellt werden, da ihre grundlegenden Auswirkungen auf das bestehende Modell aus dem Metamodell ableitbar sind. Der Systementwickler

soll somit bei der pflegenden Schema-Konsolidierung durch die Bereitstellung von **Empfehlungen** zur Behandlung von Modelländerungen unterstützt werden.

Zur systematischen Durchführung der Weiterentwicklung ist der Aufbau und Ablauf des Lösungsverfahrens der erweiterten SOM-R-Methodik in Form eines **Vorgehensmodells** zu beschreiben (Abschnitt 6.2.3).

6.2 Ein Lösungsansatz zur modellbasierten Systempflege

6.2.1 Einordnung

Der zu konzipierende Lösungsansatz zur modellbasierten Pflege von Informationssystemen wird in den Gesamtzusammenhang der modellbasierten Systementwicklung im Rahmen der vorliegenden Arbeit eingeordnet. Abbildung 6.4 stellt den grundlegenden Ablauf des Systementwicklungsprozesses grafisch dar.

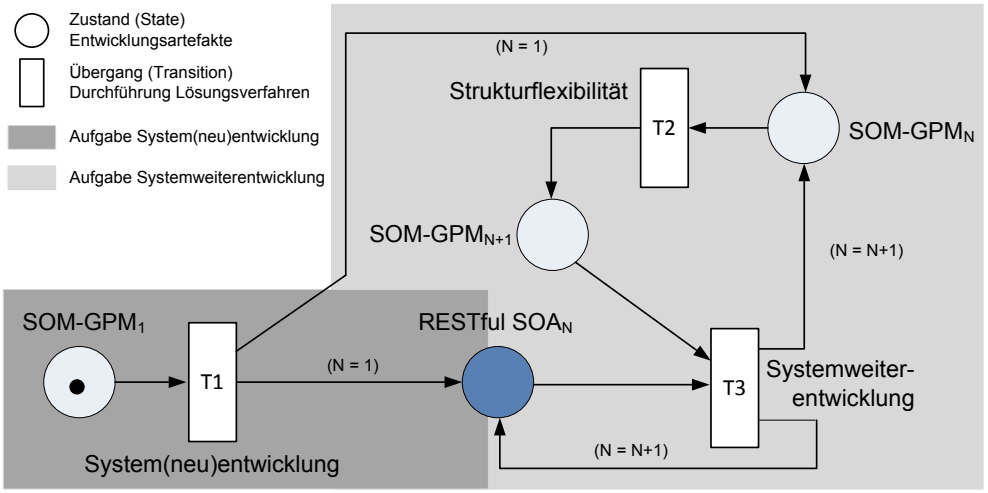


Abbildung 6.4: Gesamtüberblick über den Entwicklungsprozess

Als Darstellungsform für den Gesamtentwicklungsprozess wurde das Petri-Netz gewählt (z. B. [Reis10]). *Zustände* (States) kennzeichnen die Entwicklungsartefakte SOM-GPM und RESTful SOA, welche die Durchführung einer Entwicklungsaufgabe auslösen bzw. ihr Ergebnis darstellen. *Übergänge* (Transitions T) modellieren Lösungsverfahren, deren Durchführung mit dem „Feuern“ des jeweiligen Übergangs korrespondiert. Das Lösungsverfahren der modellbasierten Neu-Entwicklung von RESTful SOA (dunkelgrau) ist Gegenstand von Kapitel 5.

Die modellbasierte Spezifikation von RESTful SOA (Version 1) erfolgt im Rahmen der *System(neu)entwicklung* (*T1*). Der Startpunkt ist ein SOM-GPM (Version 1), welches im Petri-Netz als markierter Startzustand dargestellt wird. Die Durchführung der Weiterentwicklungsaufgabe wird ausgelöst, wenn aufgrund von *Strukturflexibilität* (*T2*) eine neue GPM-Version (N+1) vorliegt. Aufgrund der Marke in SOM-GPM$_{N+1}$ ist die Durchführung der *Systemweiterentwicklung* (*T3*) möglich, deren Ergebnis die neue Version von RESTful SOA$_N$ darstellt.

6.2.2 Bestandteile und Aufbau

Die Bestandteile und der Aufbau des Lösungsansatzes zur Pflege von Informationssystemen werden nachfolgend eingeführt und erläutert. Die erweiterte Gesamtarchitektur der SOM-R-Methodik zeigt Abbildung 6.5.

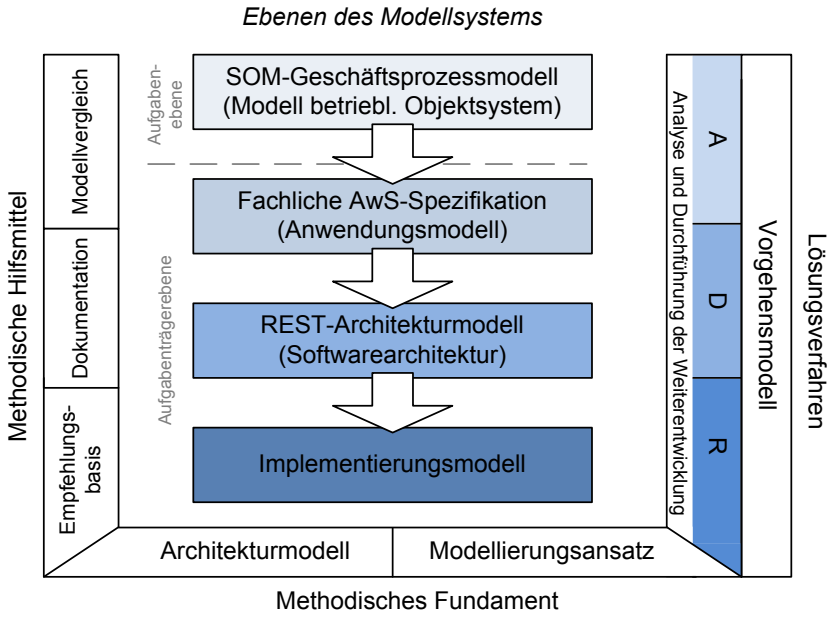


Abbildung 6.5: Bestandteile und Aufbau des Lösungsansatzes der Systempflege

Ausgehend vom strukturflexiblen SOM-Geschäftsprozessmodell wird die Weiterentwicklungsaufgabe mittels schrittweiser Anpassung der Aufgabenträgerebenen des Informationssystems durchgeführt. Die Modellierung der Aufgabenebene ist nicht Gegenstand der vorliegenden Arbeit. Die angepassten Bausteine der neuen SOM-GPM-Version bestimmen die veränderten fachlichen Anforderungen, auf die die Spezifikation der implementierten RESTful SOA abzustimmen ist. Das *methodische Fundament* bilden *Architekturmodell* und *Modellierungsansatz* der SOM-R-Methodik.

Das **Lösungsverfahren** umfasst die *Entwicklungsphasen A*, *D* und *R*, welche die Anpassung des Modellsystems entlang der Abstraktionsebenen von oben nach unten spezifizieren. Die Durchführung des Lösungsverfahrens wird mit einem *Vorgehensmodell* methodisch geleitet. Zusätzlich unterstützen folgende **methodische Hilfsmittel** die Durchführung der Entwicklungsaufgabe durch die Bereitstellung von *Assistenzinformationen*:

- *Modellvergleich* für die Identifikation von Modell-Deltas in flexiblen SOM-GPMs und auf den weiteren Ebenen des Modellsystems
- *Dokumentation* von konsolidierten Schemata und der Entwurfsentscheidungen der Schema-Konsolidierungen (K1-K3)
- *Empfehlungsbasis* zur Ableitung von Maßnahmen für die Behandlung von Veränderungen

Das angepasste Programm der RESTful SOA (Implementierungsmodell) stellt den Output des Lösungsverfahrens dar. Das überarbeitete AwS ist wieder konsistent auf die fachlichen Anforderungen des Geschäftsprozesses ausgerichtet.

6.2.3 Lösungsverfahren der Systemweiterentwicklungsaufgabe

Die Hauptschritte der Systemweiterentwicklung bilden die Phasen A, D und R, deren Durchführung durch die neue Version des Modellsystems der jeweils eingehenden Beschreibungsebene ausgelöst wird. Im weiteren Verlauf der Arbeit wird der *veraltete Stand* des Modellsystems, d.h. vor Durchführung der Anpassung an veränderte Anforderungen, als *Version N* bezeichnet. Der „*neue*“ *Stand*, nach Durchführung der Pflege, besitzt die *Version N+1*.

Die Durchführung der Hauptphasen A, D und R (*Übergänge*) wird durch eine neue Version der jeweils eingehenden Modellsysteme SOM-GPM, Anwendungsmodell und REST-Architekturmodell (*Zustände*) ausgelöst. Die Darstellung fokussiert den Ablauf der Hauptphasen und verzichtet deshalb auf die Darstellung der weiterzuentwickelnden Modellsysteme (Version N).

Das Lösungsverfahren der einzelnen Hauptphasen weist folgende logische Struktur auf:

- Jede Hauptphase gliedert sich in die Aktivitäten *Analyse* (A) und *Pflege* (P).
- Ziel der Aktivität *Analyse* ist die Erstellung eines **Überarbeitungsschemas** (Teilaktivität A2) auf den Modellebenen der Architektur. Hierzu ist die Bestimmung von Modell-Deltas (*Modelldifferenz*) in Form der Unterschiede und der Übereinstimmungen zweier Versionen der initialen Schemata notwendig (Teilaktivität A1).
  Das Überarbeitungsschema beschreibt die Vereinigung des konsolidierten $Schemas_N$ einer Modellebene mit dem aktuellen initialen $Schema_{N+1}$. Modellelemente, die entweder strukturflexibel sind oder in Beziehung mit strukturflexiblen Elementen stehen, werden als *instabil* markiert. Das Überarbeitungsschema dient als Grundlage für die Erstellung des konsolidierten $Schemas_{N+1}$.
- Die eigentliche Anpassung des SOM-R-Modellsystems erfolgt im Rahmen der Aktivität *Pflege*. Auf Grundlage der instabilen Bereiche im Überarbeitungsschema werden Handlungsempfehlungen zur Unterstützung der Schemapflege abgeleitet (Teilaktivität P1). Die pflegende Konsolidierung der geänderten Schemata wird vom Systementwickler durchgeführt (Teilaktivität P2). Ergebnis sind die konsolidierten $Schemata_{N+1}$ der überarbeiteten Modellebene.

Abbildung 6.6 zeigt das *Vorgehensmodell der erweiterten SOM-R-Methodik*, welches den Ablauf der Weiterentwicklung unter dem Blickwinkel der erstellten Artefakte darstellt. Den Aktivitäten (A, P) der Hauptphasen A, D und R werden insgesamt drei Entwicklungsstände der Schemata als Input und Output zugeordnet. Neben *initialem Schema* und *konsolidiertem Schema* wird dabei auch das *Überarbeitungsschema* als Zwischenstand im Rahmen der Weiterentwicklung erzeugt. Die Schritte zur Durchführung des Lösungsverfahrens werden in Abschnitt 6.5 detailliert erläutert.

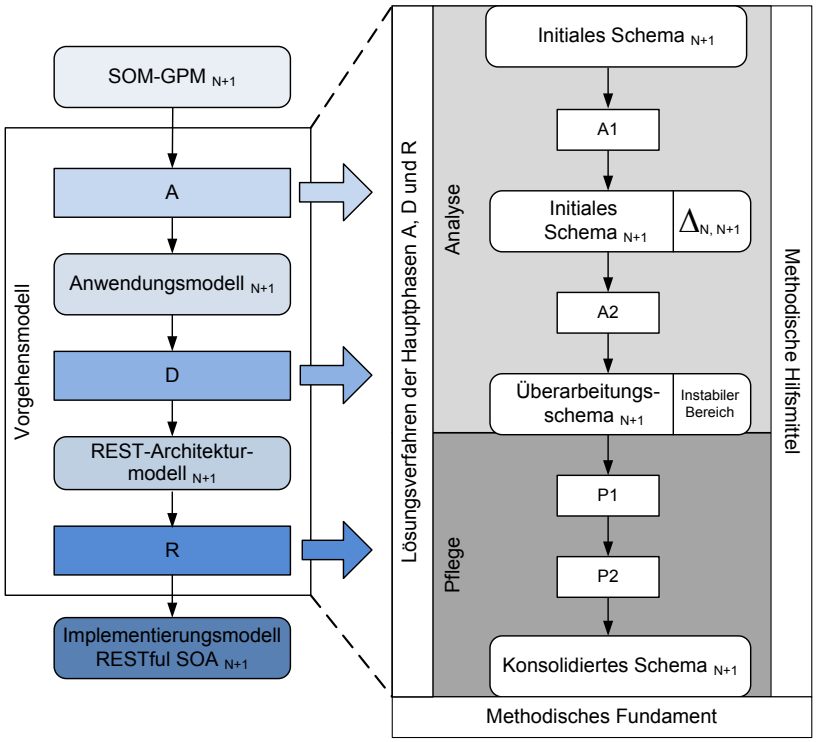


Abbildung 6.6: Erweitertes Vorgehensmodell der SOM-R-Methodik

Methodische Hilfsmittel unterstützen die Durchführung des Lösungsverfahrens in den drei Hauptphasen der Entwicklung. Die Konzeption der Hilfsmittel für *Dokumentation*, *Modell-vergleich* und *Empfehlungsermittlung* erfolgt in den Abschnitten 6.3, 6.4 und 6.5. Diese bilden zusammen mit dem Überarbeitungsschema die *Assistenzinformationen*, um den Systementwickler bei der Konsolidierung des zu pflegenden Modellsystems zu unterstützen. Auf Basis von Vorgehensmodell und methodischer Hilfsmittel erfolgt in Abschnitt 6.6 die detaillierte Gestaltung der Entwicklungsschritte zur Modellbildung auf den Abstraktionsebenen der Modellarchitektur.

6.3 Konzeption eines Dokumentationsansatzes

Modelle dienen in der modellbasierten Systementwicklung der semi-formalen Beschreibung von Anforderungen (Requirements) auf fachlicher und softwaretechnischer Ebene. Das *Requirements-Engineering* wird, als Aufgabe der Systementwicklung, durch die modellbasierte Überführung von Anforderungen auf die Aufgaben- und die fachliche Aufgabenträgerebene unterstützt (z .B. [Rupp09, S. 12 ff.], [Pohl08, S. 15 ff.]). Die *Verfolgbarkeit* bzw. *Nachvollziehbarkeit* von Anforderungen wird im Requirements-Engineering mit dem Begriff **Traceability** bezeichnet ([Rupp09, S. 397 f.], [Pohl08, S. 505 f.]). Das Ziel der Traceability-Disziplin ist, Beziehungen zwischen Anforderungen sowohl für die erstmalige Entwicklung, als auch den Prozess der Wartung und der Weiterentwicklung von Systemen zu beschreiben (z. B. [Rupp09, S. 397 ff.], [GoFi94]). Hierbei werden Verknüpfungen zwischen den dokumentierten Anforderungen erstellt, die allgemein als *Traces* oder *Traceability Links* oder auch *Relationships* bezeichnet werden ([ANR+06, S. 515 f.], [Rupp09, S. 401]).

Ziel dieses Abschnittes ist die Konzeption eines Dokumentationsansatzes, um die Verfolgbarkeit und Verfügbarkeit von Entwicklungsinformationen sicherzustellen, die während des gesamten Lebenszyklus des Informationssystems bzw. des korrespondierenden Modellsystems erzeugt werden. Die erfassten Informationen dienen zur Unterstützung der Weiterentwicklung (Analyse und Pflege) von (existierenden) SOM-R-Modellsystemen durch den Systementwickler.

6.3.1 Anforderungen an den Dokumentationsansatz der SOM-R-Methodik

Ziele der Dokumentation

Zielsetzung des zu konzipierenden Dokumentationsansatzes ist die Herstellung der Verfolgbarkeit zwischen Entwicklungsinformationen. Diese Informationen beschreiben die automatisiert und nicht-automatisiert durchführbaren Teil(schritt)e des Entwicklungsprozesses.

Folgende Anforderungen werden an den Dokumentationsansatz dieser Arbeit und die Verarbeitung der damit erfassten Informationen gestellt:

- *Einfache Repräsentationsform* der Dokumentationsbasis, um die personelle und maschinelle Verarbeitung zu unterstützen.
- Reduzierung der Abbildung auf „relevante“ Entwurfsentscheidungen, um *Nachvollziehbarkeit* zu unterstützen.
- Die Beziehungen zwischen Elementen unterschiedlicher Modellebenen sind *durchgängig* zu erfassen.

Die Forderung nach der Einfachheit der Repräsentationsform und Nachvollziehbarkeit der Entwurfsentscheidungen liegt darin begründet, dass auch die Weiterentwicklung von Modellsystemen im Kontext der modellbasierten Systementwicklung erfolgt. Die Bewältigung von Strukturflexibilität auf den Schemata einer Modellebene ist aufgrund der zu treffenden Konsolidierungsentscheidungen nur unter Einsatz des Systementwicklers durchführbar.

Bestimmung der dokumentierten Entwicklungsinformationen

Der modellbasierte Systementwicklungsprozess und seine Bestandteile sind Gegenstand („was“) des Dokumentationsansatzes. Grundsätzlich kann im Kontext der SOM-R-Methodik zwischen den erzeugten Artefakten und durchlaufenen Schritten der Modellbildung unterschieden werden (Abschnitt 5.4).

Entwicklungsartefakte der SOM-R-Methodik sind das *konsolidierte* (finale) *Schema* des *SOM-Geschäftsprozessmodells* als Ausgangspunkt der Entwicklung, sowie die *initialen Schemata* und *konsolidierten Schemata* auf den Modellebenen von Anwendungsmodell, REST-Architekturmodell und Implementierungsmodell (Abschnitte 5.2 und 5.4). Die Schemata bestehen aus Bausteinen, den Beziehungen zwischen Bausteinen sowie deren Eigenschaften.

Als **Entwicklungsschritte** zur Bildung der Schemata auf den Modellebenen wird zwischen

- *Transformationen* (T1, T2 und T3) für das automatisierte Erzeugen von Zielschemata aus Quellschemata jeweils benachbarter Modellebenen, sowie

- *Konsolidierungen* (K1, K2 und K3) zur Überarbeitung der Struktur (Bausteine, Beziehungen) und Element-Eigenschaften von Schemata einer Modellebene

differenziert (Abschnitt 5.5).

Die iterative Weiterentwicklung eines Modellsystems führt zum Entstehen einer Vielzahl von Versionen der zu speichernden Entwicklungsinformationen. Die Unterstützung einer *Versionierung* von Modellbausteinen und Beziehungen sowie der Abhängigkeiten zwischen Bausteinen, die im Rahmen der Modellerstellung gebildet werden, ist somit eine weitere Anforderung an das zu gestaltende Dokumentationskonzept. Die Speicherung der dokumentierten Elemente erfolgt in einem geeigneten Repositorium.

Bestimmung des Umfangs der Dokumentation

In Bezug auf den Umfang oder auch Grad der Dokumentation existiert eine Reihe von Alternativen, „wie“ die Speicherung von Artefakten und Entwicklungsschritten gestaltet werden kann, um das Ziel einer einfachen, nachvollziehbaren und durchgängigen Abbildung des Entwicklungsprozesses zu erreichen. Im Folgenden werden drei grundlegende Realisierungsalternativen vorgestellt:

- *Vollständige Dokumentation*: Speicherung aller Artefakte und der durchgeführten Entwicklungsschritte.
  - Die zentralen Artefakte der Dokumentation sind die *initialen* und *konsolidierten Schemata* der SOM-R-Modellebenen. Bezüglich der Durchführung von *Transformations-* und *Konsolidierungsschritten* sind für alle Operationen die Ergebnisse ihrer Ausführung zu erfassen. Hierunter fallen sowohl die Spezifikation der Modellstruktur als auch der Modelleigenschaften. Auf dieser Basis ist die Ableitung aller Zwischenstände im betrachteten Entwicklungsprozess möglich. Der erste Ansatz stellt den *Grad einer maximalen Dokumentation* dar.
  - Der *Vorteil* einer maximalen Dokumentation liegt in der vollständigen Beschreibung des Entwicklungsprozesses. Durch die Protokollierung der Ergebnisse einer Anwendung aller durchgeführten Transformations- und Konsolidierungsschritte lässt sich der Stand des Modellsystems zu jedem Entwicklungszeitpunkt herleiten.
  - Als *Nachteile* des Vorgehens stehen insbesondere der hohe Aufwand der Dokumentation und die Menge an erfassten Spezifikationsdetails dem Vorteil gegenüber (vgl. K1-K3, ab Abschnitt 5.5.3). Zudem besteht die Gefahr, dass der Umfang und die Detailfülle der erfassten Informationen für menschliche Nutzer nur schwer nachvollziehbar sind. Der Mehrwert einer maximalen Dokumentation wird auch durch die Tatsache relativiert, dass die gespeicherten Entwicklungsergebnisse auf jeder Modellebene in Form der konsolidierten Schemata beschrieben werden.
- *Dokumentation der (direkten) strukturellen Abhängigkeiten zwischen SOM-GPM und RESTful SOA:* Als zweiter Ansatz wird eine stark reduzierte, *minimale Speicherung* von Entwicklungsartefakten genauer untersucht.
  - Die zu dokumentierenden Gegenstände sind hierbei die *Schemata von SOM-GPM* und *RESTful SOA* sowie die *Abhängigkeitsbeziehungen* zwischen ihren Bausteinen. Die Dokumentation ist in diesem Fall auf das „Quellmodell“ der höchsten, und das „Ziel-

modell“ der niedrigsten bearbeiteten Abstraktionsebene als Artefakte reduziert. Die Beziehungen zwischen den Bausteinen beider Modelle werden für die durchgeführten Ableitungs- sowie Überarbeitungsschritte dokumentiert und damit die Entwicklung der Modellstruktur durchgängig beschrieben. Die spezifizierten Modelleigenschaften werden nur auf Ebene der RESTful SOA (z. B. REST-Architekturmodell) erfasst. Diese zweite Alternative kennzeichnet einen *minimalen Dokumentationsgrad*.

- *Vorteil* der minimalen Dokumentation ist die Einfachheit ihrer Realisierung durch die Beschränkung auf den Startpunkt und das Ergebnis der Systementwicklung. Für jedes Schemaobjekt (nicht seine Eigenschaften) der RESTful SOA sind die Existenzabhängigkeiten sowie die getroffenen strukturellen Entwurfsentscheidungen bis zur Ebene des SOM-GPM durchgängig beschrieben.
- *Nachteil* des Vorgehens ist die reduzierte Nachvollziehbarkeit des Entwicklungsvorgehens, da Objekteigenschaften, die die Basis für eine Vielzahl struktureller Überarbeitungen sind, nicht auf den Zwischenebenen gespeichert werden. Zudem werden Entwurfsentscheidungen der Zwischenebenen, z. B. der Ebene der fachlichen AwS-Spezifikation, nicht erfasst.

- *Dokumentation struktureller Abhängigkeitsbeziehungen zwischen Entwicklungsartefakten:* Als „Kompromiss“ bzw. Mischform zwischen den Ansätzen der vollständigen und minimalen Dokumentation wird ein mittlerer Grad als dritte und letzte Alternative untersucht. Der konzeptuelle Aufbau des Dokumentationsansatzes wird in Abbildung 6.7 dargestellt.
  - Die Speicherung von Artefakten ist hier jeweils auf die *initialen* und *konsolidierten Schemata* einer Modellebene beschränkt und ermöglicht so die vollständige Erfassung der Quell- und Zielschemata einer Ebene. Da die Durchführung der *Transformationsschritte* automatisiert erfolgt und lediglich die Eigenschaften von Modellelementen übernommen werden, ist eine Wiederherstellung der initialen Schemata, die aus den konsolidierten Schemata der übergeordneten Modellebene erstellt werden, anhand der Ableitungsregeln möglich. Ihre Erfassung in der Dokumentation wird deshalb als optional gekennzeichnet. Für die Schritte der *Konsolidierung* sind dagegen Entwurfsentscheidungen zu erfassen, die durch Anwendung von strukturellen Überarbeitungsschritten (z. B. Zerlegen, Zusammenfassen) entstehen. Die Nachvollziehbarkeit der Anreicherung von Objekteigenschaften in der Konsolidierung beschränkt sich auf die Speicherung der Ergebnisse im konsolidierten Schema. Die letzte beschriebene Alternative markiert einen *mittleren Dokumentationsgrad*.
  - *Vorteil* dieses mittleren Grades ist die Realisierung einer durchgängigen Dokumentation durch das Erfassen der strukturellen Abhängigkeiten zwischen Bausteinen und Beziehungen für alle Schemata des Architekturmodells. Die vollständige Speicherung der konsolidierten Schemata erhöht zudem die Nachvollziehbarkeit des Entwicklungsprozesses, da auch das Ergebnis der Eigenschaftenkonsolidierung gespeichert wird.
  - Als *Nachteil* ist ein erhöhter Dokumentationsaufwand für die konsolidierten Schemata von Anwendungs- und REST-Architekturmodell sowie die fehlende Erfassung von Entwicklungsentscheidungen bezüglich der Spezifikation von Modelleigenschaften zu nennen.

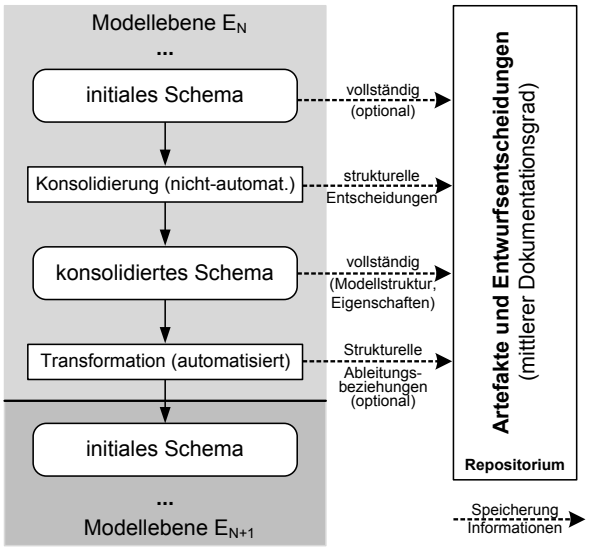


Abbildung 6.7: Grundkonzept des Dokumentationsansatzes (mittlerer Grad)

Für die Konzeption des Dokumentationsansatzes in der vorliegenden Arbeit wird die Alternative des **mittleren Dokumentationsgrades** gewählt. Sie beschreibt einen Kompromiss zwischen der möglichst weitgehenden Erfassung von Entwicklungsinformationen (hohe Nachvollziehbarkeit) und der Reduzierung auf wesentliche Entwurfsentscheidungen (Einfachheit). Die Verfolgbarkeit der Durchführung der nicht-automatisierbaren Entwicklungsschritte wird durch die Dokumentation von Abhängigkeitsbeziehungen zwischen den transformierten Bausteinen und den beiden zentralen Entwicklungszuständen jeder Modellebene gewährleistet.
Neben den Konsolidierungsbeziehungen und konsolidierten Schemata werden in der vorliegenden Arbeit auch die initialen Schemata und Transformationsbeziehungen explizit erfasst, da hiermit die Analyse der Abhängigkeiten zwischen Schemaobjekten im Modellsystem unterstützt wird. Durch die Dokumentation der initialen Schemata liegen die beiden zentralen Entwicklungsstände (initial, konsolidiert) einer Modellebene im Repositorium vor. Die initialen Schemata bilden darüber hinaus den Input für die Durchführung von Modellvergleichen (Abschnitt 6.4).

EINDEUTIGE BEZEICHNUNG DER DOKUMENTIERTEN GEGENSTÄNDE

Die Dokumentation von Artefakten und Entwurfsentscheidungen setzt die Verwendung von *Identifikatoren* voraus, damit Modellbausteine im Entwicklungsprozess der SOM-R-Methodik *global eindeutig* benannt werden können. Hierzu ist ein Ansatz erforderlich, um die

- Schemabausteine auf den Ebenen des SOM-R-Architekturmodells, die
- Ergebnisse der Modellbildung (Konsolidierung oder Transformation), sowie die
- unterschiedlichen Versionen eines Modellsystems und seiner Bausteine

identifizieren und dokumentieren zu können. Die modellierten Bausteine und die verschiedenen Entwicklungsstände des Modellsystems sollen demnach für den *gesamten Lebenszyklus* der entwickelten Systems sowie seiner iterativen Weiterentwicklung erfasst werden.

Als **eindeutiges Merkmal** wird in der vorliegenden Arbeit die Kombination aus der Typbezeichnung des *Metaobjekts* und dem *Objektnamen* des modellierten Objekts gewählt und nach dem Schema

Metaobjekt:Objektname[.Version]

gespeichert. Die Angabe einer Version erfolgt für den Fall, dass es mehrere Versionen eines Bausteins mit demselben Objektnamen gibt. Sie ist deshalb als optional ([ ]) gekennzeichnet.

Der Nutzen von *sprechenden Bezeichnern* wird in dieser Arbeit in der Förderung einer einfachen Lesbarkeit und Nachvollziehbarkeit der Informationen gesehen, die dem personellen Systementwickler auf Basis des Dokumentationskonzepts bereitgestellt werden. Als Beispiel hierfür sei ein Referenzieren der Aufgabe „empfange Auftrag“ (>Auftrag) im VES über den Identifikator „Aufgabe:>Auftrag“ genannt. Alternativ bietet sich auch die Vergabe von *Objekt-IDs* an, wie es im Falle der Implementierung einer maschinellen Verarbeitung der dokumentierten Informationen praktikabel wäre (z. B. [ANR+06, S. 523], [KDP+09, S. 2 f.]). Im Fokus dieser Forschungsarbeit steht jedoch die modellbasierte Überführung der Konzepte und Semantik aus SOM-GPMs in die RESTful SOA, welche sich im Bezeichner widerspiegeln. Im weiteren Verlauf der Arbeit werden deshalb sprechende Bezeichner verwendet.

6.3.2 Dokumentation von Modellen und strukturellen Abhängigkeiten

Die formale Beschreibung der zu dokumentierenden Entwicklungsinformationen wird nachfolgend genauer betrachtet und ein Ansatz zur Bildung von Identifikatoren sowie zur Dokumentation von Modellen und Traces vorgeschlagen.

6.3.2.1 Bildung eindeutiger Identifikatoren

Grundlage für die Realisierung des Dokumentationskonzeptes ist eine eindeutige Identifikation der modellierten Schemaobjekte auf den Ebenen der Modellarchitektur. Wie in Abschnitt 6.3.1 dargelegt, wird eine Identifizierbarkeit mittels Zuordnung des eindeutigen Bezeichners umgesetzt. Die Gegenstände der Dokumentation sind in der SOM-R-Methodik die Metaobjekte der Ebenen SOM-Geschäftsprozessmodell, fachliche AwS-Spezifikation und REST-Architekturmodell. Den Lösungsvorschlag zur Bildung der Identifikatoren fassen Tabelle 6.1, Tabelle 6.2 und Tabelle 6.3 zusammen und verdeutlichen diese an einem Beispiel.

Metaobjekt [56]	**Kürzel**	**Beispiel**
Betriebliches Objekt	bO	bO:Kunde
Betriebliche Transaktion	bT	bT:Buchung
Leistung	L	L:Flugticketverkauf
Aufgabe	A	A:>Buchung
Objektinternes Ereignis (Start_A, Ziel_A)	iE	iE:(A:>Buchung, A:Reservierung>)
Umwelt-Ereignis ([Häufigkeit,] ({Aufgabe}$^{+}$))	UE	UE:(ˈEinmal täglichˈ, (A:Werbung>))

Tabelle 6.1: Metaobjekte der *SOM-Geschäftsprozessmodellebene*

[56] Verwendete BMF-Notation (Backus-Naur-Form) zur Beschreibung von Beziehungen:
{ }$^{+}$: Wiederholung (1, *) und [ ] : Optional.

Die textuelle Notation der Metaobjekte der *SOM-Geschäftsprozessebene* zeigt Tabelle 6.1. Interne Ereignisse besitzen keinen eigenen Bezeichner und werden durch die explizite Angabe der beiden verknüpften Aufgaben identifiziert. Der Fall, dass mehrere Ereignisse dieselben beiden Aufgaben verknüpfen, wird nicht betrachtet. Zur Bildung eindeutiger Bezeichner wird für externe Ereignisse das Merkmal der Häufigkeit ihres Eintretens und die Menge der (ein bis beliebig vielen) Aufgaben mit denen diese in Beziehung stehen, festgelegt.

Auf der Ebene der *fachlichen AwS-Spezifikation* wird zwischen konzeptuellen Objekttypen, Vorgangs- und Interfaceobjekttypen sowie den Beziehungen zwischen Objekttypen unterschieden (Tabelle 6.2). *Konzeptuelle Objekttypen* können weiter als leistungs-, objekt- und transaktionsspezifische Objekttypen spezialisiert werden und damit die Bedeutungen aus der Geschäftsprozessebene (teilweise) in ihren Identifikator übernehmen.

Metaobjekt	**Kürzel**	**Beispiel**
Konzeptueller Objekttyp	KOT	KOT:Auftragskopf
Objektspezifischer Objekttyp	OOT	OOT:Kunde
Leistungspezifischer Objekttyp	LOT	LOT:Flugticketverkauf
Transaktionsspezifischer Objekttyp	TOT	TOT:Buchung
Vorgangsobjektschema	VOS	VOS:Vertrieb
Vorgangsobjekttyp	VOT	VOT:>Buchung
Interfaceobjekttyp	IOT	IOT:MCKE_Buchung
Beziehung:	---	
interacts_with (Start_OT, Ziel_OT)	---	interacts_with(TOT:Buchung, TOT:Reserv.)
is_a (Generalisierung_OT, Spezialisierung_OT)	---	is_a(KOT:Person, OOT:Kunde)
is_part_of (Ganzes_OT, Teil_OT)	---	is_part_of(KOT:Flug, KOT:Sitzplatz)

Tabelle 6.2: Metaobjekte der Modellebene *fachliche AwS-Spezifikation*

In diesem Zusammenhang wird folgende Konvention getroffen: Objekttypen, die aus dem IAS abgeleitet werden, erhalten das spezifische Kürzel (TOT, OOT, LOT) korrespondierend zu ihrem Ursprungstypen (bT, bO, L). Objekttypen des KOS, deren Bildung im Zuge der Konsolidierung erfolgt, werden mit dem Kürzel KOT versehen. Das Metaobjekt *Beziehung* ist entweder eine interacts_with-, is_a- oder is_part_of-Beziehung. Da Beziehungen keinen eigenen Namen besitzen, werden diese durch Zuordnung der verknüpften Objekttypen identifiziert. Das *Vorgangsobjektschema* ist Quelle für die Identifikation von Vorgangsdiensten im REST-Architekturmodell und wird aus diesem Grund als eigenständiges Objekt erfasst.

Das *REST-Architekturmodell* umfasst im Kern die Bausteine Objekttyp, Ressource, Datenobjekttyp und Dokumenttyp, die entsprechend dem REST-Modellierungsansatz spezialisiert werden (vgl. Abbildung 5.6). Die Bildung von Identifikatoren auf der Ebene des REST-Architekturmodells fasst Tabelle 6.3 zusammen.

Die Beziehungen zwischen Datenobjekttypen werden auf Basis der Ausführungen zur Entwicklung des REST-Architekturmodells (Abschnitt 5.5.5.4) aus den Beziehungen der Entitätsobjekttypen abgeleitet. Datenobjekttypen können als E-, ER- und R-Typen spezialisiert werden und dadurch weitere Semantik in den Bezeichner übertragen.

Metaobjekt	Kürzel	Beispiel
Entitätsspezifischer Objekttyp	EOT	EOT:Kunde
Elementarer Vorgangsobjekttyp	eVOT	eVOT:>Buchung
Nicht-elementarer Vorgangsobjekttyp	neVOT	neVOT:Versandabwicklung
Objektschema der Vorgangsdienste	OS-VD	OS-VD:Vertrieb
Interfaceobjekttyp (REST-Ebene)	R-IOT	R-IOT:MCKE_Buchung
Entitätsspezifische Ressource	ER	ER:Kunde
Elementare Vorgangsressource	eVR	eVR:>Buchung
Nicht-elementare Vorgangsressource	neVR	neVR:Versandabwicklung
Datenobjekttyp (E-, ER-, R_Typ)	DOT (E/ER/R)	E:Kunde (alternativ: DOT:Kunde)
Dokumenttyp	DT	DT:Buchung(Auftrag)
Beziehung zwischen Objekttypen:	---	
interacts_with (Start_OT, Ziel_OT)	---	interacts_with(EOT:Flugplan,EOT:Flug)
is_a (General._OT, Spezial._OT)	---	is_a(EOT:Person, EOT:Kunde)
is_part_of (Ganzes_OT, Teil_OT)	---	is_part_of(EOT:Flug, EOT:Sitzplatz)
Beziehung zwischen DOTs:	---	
Beziehung (Start_DOT, Ziel_DOT)	---	Beziehung(E:Flugplan, ER:Flug)
Generalisierung ($DOT_{gen.}$, $DOT_{spez.}$)	---	Generalisierung(E:Person, ER:Kunde)

Tabelle 6.3: Metaobjekte der *REST-Architekturmodellebene*

6.3.2.2 Modellierung von Traces

GRUNDKONZEPT DER DOKUMENTATION STRUKTURELLER ABHÄNGIGKEITSBEZIEHUNGEN

Im Fokus der Dokumentation von strukturellen Abhängigkeitsbeziehungen stehen die nicht-automatisierbaren, personell getroffenen Entwurfsentscheidungen der Konsolidierung sowie die automatisiert erstellten Abbildungsbeziehungen der Transformation. Die Abhängigkeiten zwischen Modellbausteinen werden dabei als paarweise Beziehung nach dem Schema „Baustein des Quellmodells zu Baustein des Zielmodells“ gespeichert und spezifizieren einen *Trace* oder *Traceability Link* (Abschnitt 6.3).

Besteht die Abhängigkeitsbeziehung zwischen zwei oder mehreren Bausteinen innerhalb einer Modellebene, so wird diese als **Konsolidierungsbeziehung** (*strukturelle Überarbeitungsbeziehung)* (→)[57] bezeichnet. Für die Überarbeitung der Schemastruktur sind folgende Operationen definiert (vgl. K1 und K2 der Abschnitte 5.5.3 und 5.5.5):

- *Entfernen*
- *Zerlegen* (Bilden von *Aggregationen* sowie *Generalisierungen*)
- *Zusammenfassen*

[57] Eine symbolische Unterscheidung zwischen beiden Abhängigkeitstypen wird zur visuellen Hervorhebung des dokumentierten Schrittes angewandt. Sie ist jedoch nicht zwingend erforderlich, da sie sich aus der Bezeichnung der verknüpften Metaobjekte ableiten lässt. Der Doppelpfeil (⇒) symbolisiert den Übergang zwischen zwei Abstraktionsebenen. Der Einfachpfeil (→) steht für die Überarbeitung der Schemastruktur innerhalb einer Ebene.

Nicht explizit definiert sind bisher die Operationen zum

- *Übernehmen (kein Anpassen)* und
- *Umbenennen*

von Bausteinen in den Entwicklungsschritten. Diese beiden „Operationen“ besitzen zwar keine verändernden Auswirkungen auf den strukturellen Aufbau eines Schemas, wirken sich jedoch auf die nachfolgenden Entwicklungsschritte aus. Für die vollständige Abbildung von Entwurfsentscheidungen werden sie deshalb ebenfalls untersucht.

Verknüpft eine Abhängigkeitsbeziehung zwei oder mehrere Modellbausteine auf unterschiedlichen Abstraktionsebenen, so besteht eine **Transformationsbeziehung** (*Ableitungsbeziehung*) ($\Rightarrow$)[57]. In dem Fall, dass keine vollständige Transformation des Quellschemas erfolgt, dient die explizite Erfassung von Transformationsbeziehungen der Dokumentation von Entscheidungen bezüglich einer Wahl des zu automatisierenden Teilsystems.

TRACEABILITY-METAMODELL

Die Konsolidierungs- und Transformationsbeziehungen werden in der vorliegenden Arbeit modellbasiert dokumentiert. Im Kontext der vorliegenden Arbeit wird der Begriff *Traceability* für das Metamodell des Dokumentationsansatzes verwendet, da die Nachverfolgbarkeit von Anforderungen innerhalb und zwischen Bausteinen der Modellebenen als verfolgte Zielsetzung zu gewährleisten ist. Das Traceability-Metamodell der SOM-R-Methodik definiert die verfügbaren Bausteine und die Regeln zur Modellierung struktureller Abhängigkeitsbeziehungen. Die erstellten Einträge werden als *Traces* bezeichnet. Abbildung 6.8 veranschaulicht das Traceability-Metamodell.

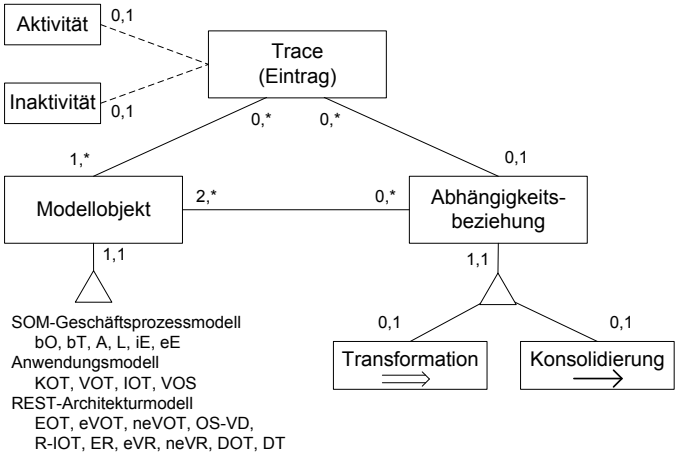


Abbildung 6.8: Traceability-Metamodell

Ein *Trace* (Eintrag) besteht aus ein bis beliebig vielen Modellobjekten, die durch eine strukturelle *Abhängigkeitsbeziehung* verknüpft sind und zueinander in einer Vorgänger-/Nachfolgerbeziehung stehen (Ausnahme: Operation Entfernen). Die Gültigkeit von Traces ist durch die Eigenschaften *Aktivität* und *Inaktivität* genauer spezifizierbar (Abschnitt 6.3.3). Eine *Abhängigkeitsbeziehung* ist entweder eine *Transformations*- oder *Konsolidierungsbeziehung*. Modellobjekte sind von einem bestimmten Typ und korrespondieren mit den modellierten

Bausteinen auf den Modellebenen des SOM-R-Architekturmodells. Modellobjekte können durch beliebig viele Beziehungen verknüpft werden.

Auch in der verwandten Literatur zur Requirements Traceability werden Metamodelle für die Modellierung von Traces vorgeschlagen (z. B. [Dick02], [Lete02]). Jedoch werden in den betrachteten Arbeiten entweder nur eine informale Verknüpfung von textuellen Anforderungen vorgestellt [Dick02, S. 7] oder Informationen der Softwareentwicklung dokumentiert, die mit dem Begriffssystem und Betrachtungsgegenstand der SOM-R-Methodik nicht übereinstimmen [Lete02, S. 32]. Diese Modellierungsansätze sind deshalb nicht, oder nur eingeschränkt, in das Modellsystem der vorliegenden Arbeit integrierbar.

Neben dem Metamodell leiten *Modellierungsregeln* die Bildung von Traceabilitymodellen in der SOM-R-Methodik im Hinblick auf die Nachbildung struktureller Operationen und definierter Abhängigkeiten zwischen Bausteinen an. Für die Visualisierung der Einträge ist weiter eine geeignete Darstellungsform zu bestimmen.

MODELLIERUNGSREGELN

1. *Definition der erlaubten Abhängigkeitsbeziehungen*: Das Traceability-Metamodell spezifiziert jeden Trace als Abhängigkeitsbeziehung zwischen einer Menge von Bausteinen, die in einer Vorgänger-/Nachfolgerbeziehung stehen. Die hierbei theoretisch möglichen Kombinationen von Abhängigkeiten werden durch die definierten Operationen der Entwicklungsschritte eingeschränkt (vgl. T1 bis K2 in Abschnitt 5.5). Abbildung 6.9 fasst erlaubte Kombinationen zwischen Objekten des SOM-R-Modellsystems zusammen und gibt die Entwicklungsschritte an, in denen die strukturellen Operationen spezifiziert sind. Eine Darstellung der Objekte, welche Beziehungen (z. B. interacts_with) repräsentieren, erfolgt nicht.

von / nach	bT	bA	bO	L	KOT	VOT	IOT	VOS	EOT	eVOT	neVOT	OS-VD	R-IOT	ER	eVR	neVR	DT	DOT
KOT	T1		T1	T1	K1.1													
VOT		T1				K1.2												
IOT						K1.2	K1.2											
VOS			T1															
EOT					T2.1				K2.1									
OS-VD								T2.1										
eVOT						T2.1				K2.2								
neVOT										K2.2		K2.2						
R-IOT							T2.1			K2.2	K2.2		K2.2					
ER					T2.2				K2.1					K2.1				
eVR						T2.2				K2.2					K2.2			
neVR											K2.2							
DT					T2.2									K2.3	K2.3	K2.3	K2.3	
DOT									K2.4									K2.4
						Transformation T1				Konsolidierung K1			Transformation T2			Konsolidierung K2		

Abbildung 6.9: Zusammenfassung der Abhängigkeitsbeziehungen zwischen Metaobjekten

Strukturelle Abhängigkeitsbeziehungen der ersten Konsolidierung K1 (Füllfarbe Lila) sind z. B. für die Kombination KOT zu KOT sowie VOT zu VOT und IOT definiert. So ist für KOTs des initialen Schemas beispielsweise das Zerlegen oder Zusammenfassen in den Überarbei-

tungsschritten von K1.1 spezifiziert. VOTs können ebenfalls zusammengefasst und ihnen IOTs zur Kommunikation mit externen Aufgabenträgern zugeordnet werden (K1.2).

2. *Dokumentation von strukturellen Entwurfsoperationen:*

   - *Entfernen*: Das Entfernen von Modellobjekten entspricht der Entscheidung für eine *Nicht-Automatisierung*. Die Anwendung der entsprechenden Operation wird in dem SOM-R-Dokumentationsansatz durch Markierung des betroffenen Bausteins repräsentiert (mittels durchstreichen z. B. KOT:Vertrieb → ~~KOT:Vertrieb~~).

   - *Zerlegen*: Modellobjekte werden zum Aufdecken von Teil-Ganzes- oder Generalisierungsbeziehungen im Rahmen der Schema-Konsolidierung zerlegt. Die Vorgängermenge umfasst das zu zerlegende Objekt und steht auf der linken Seite der Abhängigkeitsbeziehung. Die Nachfolgermenge besteht aus den erzeugten Objekten und ordnet diese auf der rechten Seite an (z. B. KOT:Auftrag → KOT:Auftragskopf, KOT:Auftragsposition).
     Eine Unterscheidung zwischen Generalisierung und Aggregation wird an dieser Stelle nicht vorgenommen, da die Beziehungen zwischen den erzeugten Modellobjekten im konsolidierten Schema dokumentiert sind. Eine explizite Darstellung der Zerlegungsart wäre beispielsweise durch die visuelle Kennzeichnung der Abhängigkeitsbeziehung (z. B. $\rightarrow^{G}$ und $\rightarrow^{A}$) möglich.

   - *Zusammenfassen*: Als weitere Operation ist das Zusammenfassen von Modellobjekten in den Regeln der Schema-Konsolidierung spezifiziert. Zusammenzufassende Bausteine werden als Vorgängermenge links, und das Ergebnis der Operation als Nachfolgermenge rechts von der Abhängigkeitsbeziehung angeordnet (z. B. KOT:Angebot, KOT:Bestellung → KOT:Auftrag).

3. *Dokumentation der weiteren Operationen*:

   - *Übernahme (keine Anpassung)*: Werden Modellbausteine im Zuge der Durchführung eines Entwicklungsschrittes nicht angepasst bzw. strukturell unverändert in ein Zielschema übernommen, so ist dies im Rahmen der Konsolidierung ebenfalls als Entwurfsentscheidung interpretierbar. Der Nachteil einer expliziten Dokumentation dieser Operation/Entscheidung wäre die Speicherung einer Vielzahl von Einträgen, die keinen Einfluss auf den Aufbau des überarbeiteten Schemas besitzen. Eine Dokumentation könnte demnach grundsätzlich entfallen, da das Entfernen von Modellobjekten bereits erfasst wird. Um die Nachvollziehbarkeit der Schemaüberarbeitung zu erhöhen, wird in den Beispielen dieser Arbeit die Übernahme von Objekten (z. B. KOT:Kunde → KOT:Kunde) explizit dargestellt.

   - Das *Umbenennen* von Modellobjekten in den SOM-R-Entwicklungsschritten wird in dieser Arbeit als Aktualisierung von Eigenschaften interpretiert. Da die Objektnamen als Teil der eindeutigen Bezeichner verwendet werden, ist eine explizite Erfassung dieser Operation notwendig. Die Darstellung könnte z. B. in Form einer Abhängigkeitsbeziehung erfolgen, deren Vorgängermenge aus dem ursprünglichen, und deren Nachfolgermenge aus dem neuen Identifikator besteht (z. B. KOT:Auftrag → KOT:Buchung).

4. *Ableitungsoperationen*: Die strukturellen Operationen der Transformationen definieren die Überführung von Modellobjekten zwischen Schemata unterschiedlicher Modellebenen. Es wird zwischen zwei grundlegende Arten unterschieden:
   - *1:1-Abbildung* (z. B. bT:Kunde ⇒ KOT:Kunde)
   - *1:N-Abbildung (Zerlegen)* (z. B. KOT:Kunde ⇒ EOT:Kunde, ER:Kunde, ER:Kundenliste)
5. *Eigenschaften von Traces*: Die Eigenschaften *Aktivität* und *Inaktivität* dienen der Versionierung von Traces. Die Aktivitätseigenschaft zeigt, zu welchem aktuellen Entwicklungszeitpunkt ein Trace erstellt wurde (*aktiv)*, oder ob er ab einem bestimmten Zeitpunkt nicht mehr gültig ist (*inaktiv)*. Die Eigenschaften spezifizieren die Erstellzeitpunkte von Dokumentationseinträgen und entsprechen damit der Schemaversion bzw. Iterationszählung des Systementwicklungsprozesses.

Grundsätzlich ist das Anpassen der bereitgestellten Traceability-Funktionalität an die jeweiligen Ziele oder Anforderungen der Systementwicklungsaufgabe möglich. Denkbar ist beispielsweise eine Verfeinerung der dokumentierten Informationen durch Kennzeichnung der Art von Strukturoperationen. Bei Konsolidierungsbeziehungen könnte weiter zwischen dem Zusammenfassen sowie der Generalisierung und Aggregation (als Zerlegungsschritte) differenziert werden. Auch die Spezifikation von Operationen zur Erfassung von Objekteigenschaften ist denkbar. Eine Diskussion von Erweiterungen, die über den vorgestellten Dokumentationsgrad hinausgehen, steht jedoch nicht im Fokus der Arbeit.

Repräsentation und Bearbeitung von Traces

Traces werden im SOM-R-Dokumentationsansatz in Modellform beschrieben. Zur Unterstützung einer Repräsentation und Verarbeitung der Vor- und Nachfolgermengen in Traces werden diese als formales Modell auf Basis des Relationenmodells interpretiert.
In den Beispielen dieser Arbeit wird zur Visualisierung von Traces eine *tabellarische Darstellungsform* gewählt, da diese es erlaubt, eine Vielzahl an Abhängigkeitsbeziehungen in einfacher und menschen-lesbarer Form darzustellen.

Ein Traceability-Modell besteht aus einer Menge von Traces, die wiederum als Tupel einer Relation interpretierbar sind. Die Struktur der Relation definiert der Relationstyp T, der aus dem kartesischen Produkt der Mengen Vorgängerobjekte V, Abhängigkeitsbeziehung B sowie Nachfolgerobjekte N gebildet wird. Die Relation T ist definiert als

$$T \subseteq V \times B \times N$$

Jeder Trace ist somit ein Tupel einer dreistelligen Relation. B beinhaltet die Elemente {⇒, →, {} }. Die Mengen von V und N müssen konform zu den definierten Abhängigkeitsbeziehungen in Abbildung 6.9 gebildet werden und enthalten *Referenzen auf Modellobjekte* in Form ihrer *eindeutigen Bezeichner*. Im Falle der Operation Entfernen sind B und N leere Mengen. Ergänzend hierzu ist auch die Erweiterung der Relation um zusätzliche Eigenschaften denkbar.

Auf Basis von dokumentierten Abhängigkeitsbeziehungen können weitere Beziehungen zwischen Modellobjekten abgeleitet werden. Für die Implikation von strukturellen Abhängigkeitsbeziehungen werden *Inferenzregeln* eingeführt:

Gegeben seien die Mengen der Modellobjekte A, B und C.

- *Substitution* (aus A → B, B → C, folgt A → C) definiert das Zusammenfassen mehrerer logisch aufeinanderfolgender Abhängigkeitsbeziehungen zu dem Zweck, die Überführung der Bausteine eines Quell- in ein Zielschema als direkte Beziehung darzustellen.
- *Vorgängervereinigung* (aus A → B, A → C, folgt A → B, C) definiert das Vereinigen von Abhängigkeiten derart, dass Beziehungen, die bezüglich ihrer Vorgängermenge übereinstimmen, zusammengefasst werden. Der Einsatz der Regel bietet sich für eine sukzessive Abfolge von Entwurfsschritten auf Modellobjekten an (z. B. erst zerlegen, dann zusammenfassen in Schritt K1.1).
- Die *Nachfolgervereinigung* (aus A → C, B → C, folgt A, B → C) bildet den Fall ab, dass eine Nachfolgermenge C aus mehreren Vorgängermengen abgeleitet wird. Die Entwicklung erfolgt dabei in mehreren Schritten.

Die Herleitung weiterer Abhängigkeiten dient insbesondere einer Unterstützung der Analyse von Modellanpassungen und der vereinfachten Darstellung der getroffenen Entwurfsentscheidungen. Die vorgestellten Inferenzregeln sind dabei an die Armstrong-Axiome angelehnt, welche die Implikation von funktionalen oder mehrwertigen Abhängigkeiten aus einer gegebenen Abhängigkeitsmenge beschreiben [Arms74].

Nr	V	B	N
1	bT:Angebot	⇒	TOT:Angebot
2	bT:Bestellung	⇒	TOT:Bestellung
3	TOT:Angebot	→	KOT:Angebotskopf, KOT:Angebotspos.
4	TOT:Bestellung	→	KOT:Bestellkopf, KOT:Bestellpos.
5	KOT:Angebotskopf, KOT:Bestellkopf	→	KOT:Auftragskopf
6	KOT:Angebotspos., KOT:Bestellpos.	→	KOT:Auftragsposition
Abgeleitete Abhängigkeitsbeziehung			
A1	TOT:Angebot, TOT:Bestellung	→	KOT:Auftragskopf, KOT:Auftragspos.

Tabelle 6.4: Repräsentation von Entwurfsentscheidungen (Traces) an einem Beispiel

Die tabellarische Visualisierung und Überarbeitung von Entwurfsentscheidungen zeigt Tabelle 6.4 an einem einfachen Beispiel. Ausgehend von den betrieblichen Transaktionen *Angebot* und *Bestellung* werden konzeptuelle Objekttypen abgeleitet (T1, Nr. 1 und 2) und anschließend konsolidiert (K1, Nr. 3 bis 6). Die strukturelle Überarbeitung sieht aus Gründen der Normalisierung zunächst eine Zerlegung der beiden Objekttypen in einen *Kopf* und eine *Position* vor. Anschließend werden diese jeweils zu einem *Auftragskopf* und einer *Auftragsposition* zusammengefasst, da sie bezüglich ihrer Eigenschaften weitgehend überlappen. Auf Basis der Abhängigkeitsbeziehungen wird durch die Anwendung von Substitution und Nachfolgervereinigung eine weitere Beziehung gebildet (A1), welche die direkte Überführung der spezifizierten Bausteine von Quell- in Zielschema darstellt.

6.3.3 Verwaltung von Modellen und Traces

Im SOM-R-Dokumentationsansatz werden die Modelle und Traces, die die Ergebnisse bzw. die Entwurfsentscheidungen des auf Basis der SOM-R-Methodik durchgeführten Systement-

wicklungsprozesses repräsentieren. Im Rahmen der Weiterentwicklung können neue Einträge der Dokumentation hinzugefügt werden und/oder bestehende Einträge veralten, welche dann ab einem bestimmten Entwicklungsstand ungültig sind. Um die Wiederherstellbarkeit eines bestimmten Entwicklungsstandes des Modellsystems bzw. der damit korrespondierenden Systemversionen sicherstellen zu können, wird der Dokumentationsansatz in diesem Abschnitt um ein Vorgehen zur Versionierung der gespeicherten Modelle und Traces erweitert.

Mit dem Einsatz von eindeutigen Bezeichnern für das Referenzieren von Modellelementen sowie der Beschränkung von strukturellen Modellanpassungen auf die elementaren Änderungsoperationen (*Hinzufügen* und *Entfernen*) wurden die zentralen Rahmenbedingungen der Konzeption bereits diskutiert (Abschnitt 4.2.3). Nachfolgend werden Anforderungen an die Versionierung von Entwicklungsartefakten und -schritten betrachtet.

VERWALTUNG VON MODELLEN

Die Bestimmung von verschiedenen Versionen eines Modellsystems setzt die Definition eines Unterscheidungsmerkmals für *Entwicklungsstände bzw. -versionen der konsolidierten Schemata* voraus. Im Kontext der SOM-R-Methodik bietet sich die Verwendung der Iterationsläufe des Entwicklungsprozesses als einfaches Merkmal an (vgl. Iterationsnummern, Abbildung 6.4):

- Ein neu-entwickeltes Schema markiert die erste Systemversion und besitzt damit die Iterationsnummer N = 1.
- Jeder Durchlauf des Systemweiterentwicklungsprozesses führt zur Bildung neuer Schemaversionen auf den Modellebenen der Architektur (N = N+1).

Die Version eines Schemas wird stets durch die Anpassung seiner Bausteine und Beziehungen aktualisiert. Wie bereits ausgeführt, ist die Speicherung der Gesamtschemata (vollständiges Schema) im Rahmen des Dokumentationsansatzes ausreichend. Die Versionsverwaltung von einzelnen geänderten Modellbestandteilen wird deshalb in den nachfolgenden Ausführungen nicht untersucht.

VERWALTUNG VON TRACES

Die Verwaltung von Abhängigkeitsbeziehungen ist die zweite Aufgabe des Dokumentationsansatzes. Im Kern stehen dabei insbesondere das Erfassen und Verwalten der nicht-automatisierten Entwurfsentscheidungen, die im Zuge der Schema-Konsolidierungen durch den Entwickler getroffen werden. Dieses Vorgehen ermöglicht die durchgängige Beschreibung von Abhängigkeiten zwischen Modellbausteinen auf den Ebenen des SOM-R-Architekturmodells. Die hierbei dokumentierten Entwurfsschritte sind derart zu verwalten, dass für die verschiedenen Schemaversionen die jeweils gültige Menge an Traces bestimmt werden kann.

Die Versionierung von Traces erfolgt auf Basis der Eigenschaften *Aktivität* und *Inaktivität*.

- **Aktivität (A)** spezifiziert für jeden Trace den *Erstellzeitpunkt*, ab dem dieser aktiv im Rahmen der Schemabildung verwendet wird.
- **Inaktivität (I)** markiert den Zeitpunkt, ab dem der Eintrag zu einem Trace ungültig ist und nicht mehr bei der Schemaentwicklung angewendet wird.

Die Zeitpunkte werden folglich auf Basis der Versionsnummer einer Schemaiteration erfasst. Traces sind vom „Aktivitätszeitpunkt“ bis zum „Inaktivitätszeitpunkt – 1“ gültig.

Als Gründe für das Inaktiv-werden von Entwurfsentscheidungen lassen sich insbesondere zwei Ursachen anführen. Zum einen können referenzierte Vorgängerobjekte in einem vorausgegangenen Entwicklungsschritt entfernt bzw. nicht mehr gebildet worden sein. Zum anderen ist möglich, dass die Abhängigkeitsbeziehung selbst entfällt (z. B. aufgrund einer geänderten Entwurfsentscheidung).

Nr	V	B	N	A	I
3	TOT:Angebot	→	KOT:Angebotskopf, KOT:Angebotspos.	1	2
4	TOT:Bestellung	→	KOT:Bestellkopf, KOT:Bestellposition	1	2
5	KOT:Angebotskopf, KOT:Bestellkopf	→	KOT:Auftragskopf	1	2
6	KOT:Angebotspos., KOT:Bestellpos.	→	KOT:Auftragsposition	1	2
7	TOT:Angebot	→	~~TOT:Angebot~~	2	
8	TOT:Bestellung	→	KOT:Auftragskopf, KOT:Auftragspos.	2	

Tabelle 6.5: Verwaltung von Traces an einem Beispiel

Tabelle 6.5 veranschaulicht die Verwaltung von Traces am Beispiel der Weiterentwicklung eines KOS (siehe Tabelle 6.4). Am konsolidierten KOS in seiner ersten Version wird eine Reihe von Anpassungen vorgenommen. Der TOT *Angebot* wird entfernt (Entscheidung zur Nicht-Automatisierung) und ist somit ab der aktuellen Schemaversion (2) nicht mehr gültig und nicht mehr in der aktuellen Version des Schemas enthalten. Als Ergebnis dieser Entscheidung wird der Eintrag 3 inaktiv. Die Inaktivität pflanzt sich auf die abhängigen Einträge 5 und 6 fort. Stattdessen entscheidet sich der Entwickler für die Zerlegung der Bestellung in *Auftragskopf* und *-position* (Eintrag 8). Die Traces 7 und 7 werden demnach zur Konsolidierung der zweiten Schemaversion verwendet.

Mit der Konzeption des Dokumentationsansatzes der SOM-R-Methodik wird eine Lösung zur Speicherung von Entwurfsbeziehungen vorgeschlagen. Die Dokumentation ist auf die Beschreibung von strukturellen Abhängigkeiten zwischen Modellbausteinen innerhalb und zwischen den Ebenen des SOM-R-Architekturmodells beschränkt. Von weiteren Details der Bausteineigenschaften wird dabei abstrahiert. Die Beziehungen beschreiben alle notwendigen Existenzabhängigkeiten, um eine *Analyse* der Auswirkungen von Schemaänderungen durchführen zu können. Zudem ermöglichen sie eine Erfassung aller strukturellen Entwurfsentscheidungen, die im Rahmen der Systementwicklung getroffen werden und repräsentieren diese in einfacher Form.

Erläuterung des Dokumentationsansatzes an einem Beispiel

Der SOM-R-Dokumentationsansatz wird abschließend anhand der Fallstudie Flugbuchung erläutert. Als Bausteine des SOM-Geschäftsprozessmodells werden die betriebliche Transaktion *Angebot* sowie die Aufgaben *Angebot senden* (Kunde) und *Angebot empfangen* (Vertrieb) untersucht. Das ausgewählte Beispiel stellt eine nicht-triviale strukturelle Überarbeitung dar und umfasst eine Vielzahl an Entwurfsentscheidungen, um so die Dokumentationsregeln des vorgestellten Konzepts möglichst umfassend erläutern zu können.

Nr	V	B	N
[1]	bT:Angebot	⇒	TOT:Angebot
[2]	A:Angebot>	⇒	VOT:Angebot>
[3]	A:>Angebot	⇒	VOT:>Angebot
[4]	TOT:Angebot	→	TOT:Angebotskopf, TOT:Angebotsposition
[5]	TOT:Angebotskopf, TOT:X_kopf	→	KOT:Auftragskopf
[6]	TOT:Angebotsposition, TOT:X_position	→	KOT:Auftragsposition
[7]	VOT:Angebot>	→	VOT:Angebot>, IOT:MCKS_Angebot
[8]	VOT:>Angebot	→	~~VOT:>Angebot~~
[9]	KOT:Auftragskopf	⇒	EOT:Auftragskopf, ER:Auftragskopf, ER:Auftragskopfliste
[10]	KOT:Auftragsposition	⇒	EOT:Auftragsposition, ER:A.position, ER:Auftragspositionsliste
[11]	VOT:Angebot>	⇒	eVOT:Angebot>, eVR:Angebot>, eVR:Angebot>Liste
[12]	IOT:MCKS_Angebot	⇒	R-IOT:MCKS_Angebot
[13]	KOT:Auftragskopf, KOT:Auftragsposition	⇒	DT:Angebot(Auftrag), DT:X_(Auftrag)
[14]	EOT:Auftragskopf	→	EOT:Auftragskopf, DOT:Auftragskopf
[15]	EOT:Auftragsposition	→	EOT:Auftragsposition, DOT:Auftragspos.
[16]	ER:Auftragskopf, ER:Auftragskopfliste	→	ER:Auftragskopf, ER:Auftragskopfliste, ER:Auftragskopf-Status, ER:Stornierung, ER:Stornierungsliste
[17]	ER:Auftragsposition, ER:A.positionsliste	→	ER:Auftragsposition, ER:A.positionsliste
[18]	eVOT:Angebot>, eVOT:>Flugwahl	→	eVOT:>Angebotsabwicklung
[19]	eVR:Angebot>, eVR:>Flugwahl, eVR:Angebot>Liste, eVR:>Flugwahlliste	→	eVR:>Angebotsabwicklung eVR:>Angebotsabwicklungsliste
[20]	R-IOT:MCKS_Angebot, R-IOT:MCKE_Flugwahl	→	R-IOT:MCKS_Angebotsabwicklung

X = {Wahl, [Angebot,] Angebotsbestätigung, Auftragszusammenfassung, Buchung, Reservierung, Reservierungsbestätigung, Buchungsbestätigung}

Tabelle 6.6: Traces zur Spezifikation der Angebotserstellung in der Fallstudie (Ausschnitt)

Tabelle 6.6 zeigt die Abhängigkeitsbeziehungen, welche die modellbasierte Überführung der GP-Bausteine zur *Angebotsabwicklung* in die Komponenten der RESTful SOA dokumentieren. Die Nummerierung der Einträge beginnt aus Gründen der Lesbarkeit bei 1. Die vollständige Dokumentation der Entwicklungsschritte ist in den Tabellen der Anhänge A.3 und A.5 dargestellt. Alle Einträge sind aktiv und erfassen die Gestaltung der ersten Version eines SOM-R-Modellsystems (Aktivität: Version 1). Die Aktivität bzw. Inaktivität von Traces ist nicht dargestellt. Die zugehörigen initialen und konsolidierten Schemata entsprechen der durchgeführten Modellbildung in der Fallstudie (vgl. Abbildung 4.11, Abbildung 5.12, Abbildung 5.13, Abbildung 5.14 und Abbildung 5.15).

Der *TOT:Angebot* des initialen KOS wird aus Gründen der Normalisierung in einen *Kopf* und seine *Positionen* zerlegt und anschließend mit weiteren TOTs (vgl. S. 121, X = Auftragsstatus) zusammengefasst (Nr.4-6). Der VOT *Angebot>* wird automatisiert und ein IOT zur Kommunikation mit dem Umweltobjekt *Kunde* zugeordnet (Nr. 7). *VOT:>Angebot* wird dagegen nicht automatisiert und deshalb entfernt (Nr. 8). Die Ableitung des initialen Dokumentenschemas (T2) umfasst neben dem Dokumenttyp *Angebot* auch weitere Doku-

menttypen, welche die Zustände der Entitätsdienste *Auftragskopf* und -*position* spezifizieren (Nr.11-13). Als zusätzliche Quelle kann die Strukturdefinition von *TOT:Angebot* mit in die Ableitung einfließen. Die EOTs des initialen OS-ED werden konsolidiert und aus diesen jeweils ein DOT abgeleitet. Zudem werden die ERs des initialen Ressourcenschemas überarbeitet und verfeinert (Nr.16, 17). Im Rahmen der Konsolidierung von OS-VD und Ressourcenschema wird der elementare Vorgangsdienst *Angebot* (eVOT, eVR) mit der *Flugwahl* zusammengefasst und der Vorgangsdienst *Angebotsabwicklung* gebildet (Nr.18, 19).

Die modellbasierte Überführung der betrieblichen Transaktion *Angebot* und der korrespondierenden Aufgaben in die Komponenten der RESTful SOA sind durchgängig und die strukturellen Entwurfsentscheidungen vollständig dokumentiert. Somit ist für jeden Baustein des SOM-R-Modellsystems bestimmbar, von welchen Bausteinen dieser existenziell abhängig ist, und umgekehrt, welche Bausteine von diesem abhängig sind.

6.4 Konzeption eines Modellvergleichsansatzes

Änderungen im strukturflexiblen SOM-Geschäftsprozessmodell sind der Auslöser für die Weiterentwicklung eines existierenden Anwendungssystems RESTful SOA. Als Voraussetzung für eine zielgerichtete Anpassung der RESTful SOA ist ein Ansatz zur Bestimmung von Veränderungen auf den Ebenen des SOM-R-Architekturmodells erforderlich. Die Identifikation von Unterschieden in geänderten Schemata ist die Basis für die weitere Analyse und Anpassung des gesamten Modellsystems, so dass dessen Ebenen konsistent aufeinander abgestimmt sind. Im nachfolgenden Abschnitt wird ein *Modellvergleichsansatz* als methodisches Hilfsmittel der SOM-R-Methodik konzipiert, um damit die Durchführung der *Analyseaktivität* im Rahmen der Systemweiterentwicklung zu unterstützen.

In Abschnitt 6.4.1 werden zunächst die methodischen Rahmenbedingungen für die Durchführung von Modellvergleichen in SOM-R-Modellsystemen erarbeitet. Die Konzeption des SOM-R-Modellvergleichsansatzes und eine Betrachtung von Realisierungsalternativen erfolgt in Abschnitt 6.4.2.

6.4.1 Methodische Rahmenbedingungen

6.4.1.1 Metaebenen von Modelländerungen

Der Ursprung von Modelländerungen kann grundsätzlich auf den vier Metaebenen der Modellierung liegen ([FeSi13, S. 138], siehe auch Abschnitt 2.1.5). Im Falle von Anpassungen auf einer Metaebene N sind deren Auswirkungen auf die jeweils niedrigeren Ebenen (N-X, $X \geq 1$) zu prüfen. Beispielsweise könnten sich Änderungen des Meta-Metamodells auf die zugrundeliegenden Metamodelle und Schemata fortpflanzen.

Im Fokus der vorliegenden Arbeit steht die Unterstützung einer Bewältigung von Änderungen in strukturflexiblen SOM-Geschäftsprozessmodellen, welche durch strukturelle Anpassungen auf der *Schemaebene* entsteht. Das Meta-Metamodell der verwendeten Modellierungsansätze sowie die integrierten Metamodelle des SOM-R-Architekturmodells sind dagegen stabil und weisen keine Flexibilität auf. Abbildung 6.10 fasst diese Ausgangssituation am Beispiel struk-

turflexibler SOM-Geschäftsprozessmodelle zusammen (vgl. Metaebenen in [FeSi13, S. 138 f.]). Von den Ausprägungen auf der Instanzebene wird abstrahiert.

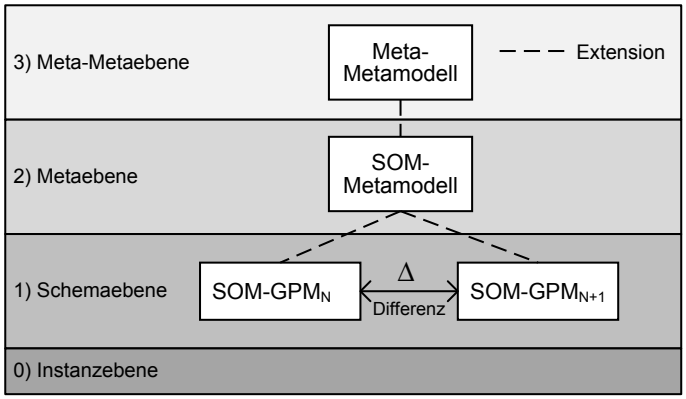


Abbildung 6.10: Relevante Änderungen auf den Metaebenen von SOM-Geschäftsprozessmodellen

Die Aufgabe des Modellvergleichsansatzes ist die Bestimmung der *Modelldifferenz* zwischen zwei Versionen (N, N+1) eines Modells der SOM-Geschäftsprozessebene, die in Form eines *Modell-Deltas (Δ)* repräsentiert wird. Grundlage für die Konzeption des Ansatzes ist die Untersuchung von Änderungsoperationen sowie Bestimmung von geänderten Modellelementen bzw. -bereichen.

6.4.1.2 Elementare Änderungsoperationen und ihre Auswirkungen auf die Schemastruktur

Strukturelle Schemaänderungen[58] werden in der SOM-R-Methodik auf Basis der elementaren Operationen *Hinzufügen* und *Entfernen* von Modellbausteinen spezifiziert ([WSL+11a, S. 90], [WRR07, S. 585], siehe auch Abschnitt 4.2.3). Das Hinzufügen/Entfernen von Modellbausteinen zieht, aufgrund der definierten wechselseitigen Abhängigkeit im Meta-Metamodell, stets eine korrespondierende Änderung in den Beziehungen der Bausteine nach sich (vgl. Abbildung 2.5).

Das Verständnis der **elementaren Änderungsoperationen** in dieser Arbeit wird nun eingeführt:

- **Operation Hinzufügen** spezifiziert das Einfügen (Symbol: '**+**') von Modellbausteinen in ein Schema, einschließlich der zugehörigen Beziehungen, um Verknüpfungen zu bestehenden Modellbausteinen herzustellen.
- **Operation Entfernen** spezifiziert das Löschen (Symbol: '**–**') von Modellbausteinen aus einem Schema, einschließlich der zugehörigen Beziehungen, um Verknüpfungen zu benachbarten Modellbausteinen zu entfernen.

Die elementaren Operationen bilden die Grundlage, um das Durchführen von Anpassungen auf den Modellebenen des SOM-R-Architekturmodells zu beschreiben. Die Anwendung der Operationen erfolgt dabei stets unter Einhaltung des zugrundeliegenden Metamodells und der Modellierungsregeln. Abbildung 6.11 zeigt die Anwendung der Änderungsoperation

[58] Im Kontext der Untersuchung von Strukturflexibilität werden in der vorliegenden Arbeit die Begriffe *Ändern* oder *Aktualisieren* von existierenden Bausteinen synonym zur Durchführung des *Entfernens* und *Hinzufügens* einer neuen Version der Schemabausteine und ihrer Beziehungen verwendet.

Hinzufügen, einmal in allgemeiner Form (links), und einmal am Beispiel der SOM-Modellierungssprache für Geschäftsprozessmodelle (rechts).

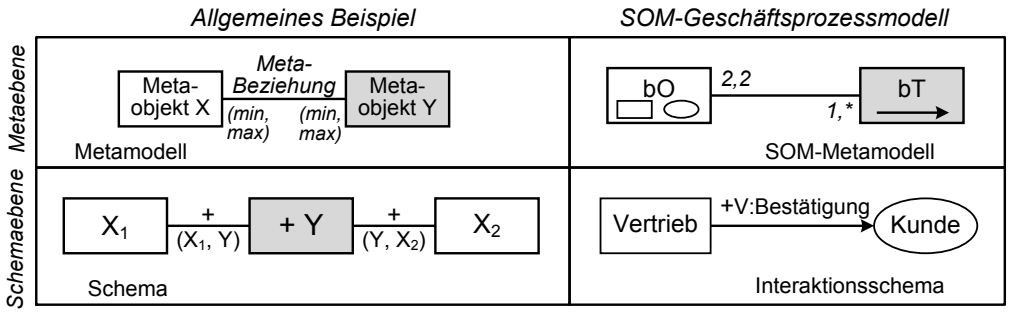


Abbildung 6.11: Auswirkungen der Änderungsoperation *Hinzufügen* an einem Beispiel

Zunächst wird das Hinzufügen eines Modellbausteins an einem generischen Beispiel betrachtet. Der eingefügte (+) Modellbaustein Y wird durch Hinzufügen zweier Beziehungen mit den bestehenden Bausteinen X_1 und X_2 verknüpft. Die Beziehung zwischen den Modellbausteinen vom Typ X und Y ist auf der Metaebene durch ein Metamodell definiert.

Der allgemein beschriebene Sachverhalt wird auf SOM-Geschäftsprozessmodelle übertragen und am Beispiel des Hinzufügens der betrieblichen Transaktion *Bestätigung* konkretisiert (rechte Seite). Die Visualisierung der Schemaebene beschränkt sich aus Gründen der einfachen Darstellung auf die hinzugefügte V-Transaktion des IAS und blendet weitere Transaktionen aus. Zudem werden die korrespondierenden Änderungen im VES nicht untersucht. Es sei an dieser Stelle noch einmal darauf hingewiesen, dass in der grafischen Darstellungsform des IAS auf die explizite Visualisierung von Beziehungen zwischen Modellbausteinen verzichtet wird. Es werden lediglich Bausteine (z. B. betriebliche Transaktionen, objektinterne Ereignisse) modelliert. Die Beziehungen zwischen Bausteinen werden ausschließlich durch die Darstellung der Verknüpfung bzw. des „Andockens“ zweier Bausteine erfasst. Dies gilt ebenfalls für alle anderen Schemata des SOM-R-Modellsystems.

Auf Basis der elementaren Änderungsoperationen lassen sich weitere **komplexe Operationen** spezifizieren. In der Literatur werden häufig *Verschieben* und *Ersetzen von Modellbausteinen* sowie das *Ändern von Bausteineigenschaften* genannt (z. B. [HRO+10, S. 5], [WRR07, S. 579], [CDP07, S. 171], [XiSt05]). Diese Operationen werden im weiteren Verlauf jedoch nicht betrachtet, da sie, wie bereits ausgeführt, als Folge von elementaren Operationen formuliert werden können. Nach diesem Verständnis wird auch ein *Ändern von Eigenschaften* als Entfernen des ursprünglichen und Hinzufügen eines neuen Bausteins sowie seiner Beziehungen zu benachbarten Modellbereichen beschrieben.

6.4.1.3 Stabile und instabile Teilbereiche von Modellsystemen

Ausgehend von den elementaren Operationen werden die Modellbausteine eines angepassten Schemas und dessen Beziehungen in die drei Gruppen eingeteilt:

1. Unveränderte Bausteine und unveränderte Baustein-Beziehungen.
2. Veränderte Bausteine und ihre Beziehungen zu benachbarten Bausteinen.

3. Unveränderte Bausteine, die mit veränderten Bausteinen in Beziehung stehen.

Die dritte Gruppe spezifiziert in einem angepassten Schema die Schnittstelle zwischen dessen unveränderten und den veränderten Modellelementen.

Die Bausteine und die zugehörigen Beziehungen der zweiten und dritten Gruppe werden im Folgenden als **instabil** und der hierdurch gebildete Ausschnitt des Modellsystems als **instabiler Teilbereich** bezeichnet. Die instabilen Bereiche eines Modells sind direkt von einer Modelländerung betroffen oder stehen mit Elementen eines veränderten Modellbereichs in Beziehung. Die unveränderten, jedoch instabilen Bausteine werden mit dem Symbol ('**i**') als *Instabilitätsmarkierung* gekennzeichnet. Der **stabile Teilbereich** einer Modellebene umfasst diejenigen unveränderten Modellbausteine, die ausschließlich mit unveränderten Bausteinen in Beziehung stehen. Der instabile und stabile Bereich überlappen sich bezüglich der als (i) markierten Elemente.

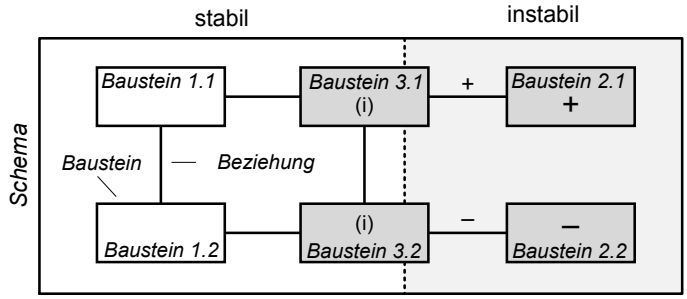


Abbildung 6.12: Stabiler und instabiler Teilbereich in einem Schema (allgemeines Beispiel)

Abbildung 6.12 veranschaulicht die Bildung der drei *Modellelementgruppen* und des stabilen und instabilen Teilbereichs in allgemeiner Form. Die Gruppe unveränderter Modellelemente enthält die Bausteine 1.1 und 1.2 sowie deren Beziehungen (Gruppe 1). Dem Schema neu hinzugefügt bzw. aus diesem entfernt wurden die Bausteine 2.1 und 2.2 und zugehörige Beziehungen (2.1-3.1 und 2.2-3.2) (Gruppe 2). Sie bilden zusammen mit den (unveränderten) Bausteinen 3.1 und 3.2 (Gruppe 3) den instabilen Modellbereich. Der stabile Modellbereich umfasst die unveränderten Bausteine 1.1, 1.2, 3.1 und 3.2 sowie die vier Beziehungen zwischen diesen Bausteinen.

6.4.2 Modellvergleichsansatz der SOM-R-Methodik

Die Bestimmung von Unterschieden und Übereinstimmungen zwischen Modellen oder Modellversionen ist Voraussetzung dafür, dass die Modellebenen der SOM-R-Architektur zielgerichtet an Veränderungen angepasst werden. Die Änderungen im strukturflexiblen SOM-Geschäftsprozessmodell sind dabei der Startpunkt des Weiterentwicklungsprozesses. Nachfolgend wird die Konzeption eines Modellvergleichsansatzes als methodisches Hilfsmittel der SOM-R-Methodik vorgestellt, auf dessen Basis *Modelldifferenzen* als Menge von veränderten sowie unveränderten Modellelementen bestimmt werden.

6.4.2.1 Bestandteile und Ablauf

Die Bestandteile und den Aufbau des Modellvergleichsansatzes sowie den schematischen Ablauf der Differenzbestimmung zeigt Abbildung 6.13. Gemäß der Problemstellung dieser Arbeit wird im Folgenden ausschließlich der Vergleich von zwei Versionen eines Modells untersucht.

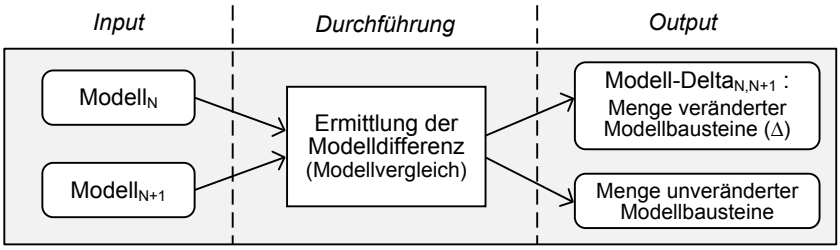


Abbildung 6.13: Bestandteile des SOM-R-Modellvergleichs

Den *Input* für die Durchführung des Modellvergleichs und damit die Ermittlung der Modelldifferenz bilden die beiden Versionen eines Modells ($Modell_N$ und $Modell_{N+1}$). *Output* des Modellvergleichs sind die *Menge unveränderter Modellbausteine* (und unveränderter Baustein-Beziehungen), die in beiden Modellversionen übereinstimmen und somit identisch sind, sowie das *Modell-Delta* (Symbol: '**Δ**'). Das Modell-$Delta_{N,N+1}$ spezifiziert die Menge der hinzugefügten und/oder entfernten Bausteine und Beziehungen des $Modells_{N+1}$ im Vergleich zu $Modell_N$.

Die ermittelten Ergebnisse dienen im Rahmen der Analyseaktivität zur Bestimmung der Auswirkungen von Änderungen auf das existierende Modellsystem und der Markierung von instabilen Bereichen in den Überarbeitungsschemata des Weiterentwicklungsprozesses (siehe auch Abschnitte 6.2.3 und 6.6.1).

Nachfolgend werden Lösungsansätze zur

- *Ermittlung von Modelldifferenzen* (Durchführung des Modellvergleichs) sowie der
- *Spezifikation und Repräsentation* des Modell-Deltas (Output)

für den Modellvergleichsansatz der SOM-R-Methodik konzipiert. Die Ausgestaltung des SOM-R-Modellvergleichsansatzes erfolgt stets unter Einhaltung der methodischen Rahmenbedingungen aus Abschnitt 6.4.1. Die Definition einer geeigneten Repräsentationsform soll die menschliche und die maschinelle Verarbeitung der ermittelten Modelldifferenz unterstützen.

Für die Gestaltung von Modellvergleichsansätzen und die Bestimmung von Modelldifferenzen schlagen BRUN ET AL [BrPi08] eine allgemeine Gliederung des Vergleichsvorgehens in den Aktivitäten Berechnung (*calculation*), Repräsentation (*representation*) und Visualisierung (*visualization*) vor [BrPi08, S. 30 f.][59]. Auch wenn den Ausführungen der Autoren das Projekt *EMF Compare* als konkrete Problemstellung zugrunde liegt [EMFC14], so sind die darin

[59] Das in [BrPi08] vorgestellte Lösungsverfahren spezifiziert die Durchführung von Modellvergleichen mit dem Modellierungswerkzeug *EMF Compare* (Webseite: http://www.eclipse.org/emf/compare/, Abruf am 2. März 2015).

vorgeschlagene Vorgehensweise und Bestandteile eines Vergleichsansatzes grundsätzlich auch zur Strukturierung der weiteren Konzeption in dieser Arbeit geeignet.

Die Durchführung von Modellvergleichen ist in der vorliegenden Arbeit auf die Ermittlung des Hinzufügens und Entfernens von Bausteinen bzw. Objekten beschränkt (Abschnitt 6.4.1.2). Die Anpassung von Schemaobjekten zieht dabei stets eine *korrespondierende Änderung ihrer Beziehungen* zu benachbarten Objekten nach sich. Demzufolge können Beziehungen nicht unabhängig von den verknüpften Objekten verändert werden (siehe Abbildung 2.5). Im weiteren Verlauf der Arbeit wird primär von einem Vergleich der Bausteine bzw. Objekte des Modells gesprochen und auf die zusätzliche Angabe der abhängigen Beziehungen verzichtet.

6.4.2.2 Ermittlung von Modelldifferenzen

6.4.2.2.1 Einführung

Das *Ziel der Differenzermittlung* ist die Bestimmung von Unterschieden zwischen den Bausteinen zweier (oder auch mehrerer) Modelle. Die paarweise Übereinstimmung von zwei Modellbausteinen und ihrer Beziehungen wird dabei als *Korrespondenz* bezeichnet [Kelt07, S. 2]. Die Menge an Korrespondenzen umfasst somit die Modellbausteine, welche in zwei Versionen eines Modells gemeinsam vorkommen. Ob sie gemeinsam vorkommen, also gleich oder ähnlich sind, wird durch Anwendung eines geeigneten *Vergleichsverfahrens* bestimmt. Neben einer Menge übereinstimmender Modellbausteine können die verglichenen Modelle eine Menge ungleicher Elemente besitzen. Allgemein sind dies eingefügte, gelöschte oder veränderte Bausteine einer Modellversion (z. B. [CDP07, S. 170], [TBW+07, S.298]). Da ein Lösungsverfahren zur Durchführung von Modellvergleichen normalerweise in algorithmischer Form spezifiziert ist, wird diese Phase synonym zum Begriff der *Differenzberechnung* verwendet ([BrPi08, S. 30], [Kelt07, S. 1]).

Die Vergleichsverfahren zur Ermittlung von Modelldifferenzen werden in der Literatur anhand des eingesetzten Vergleichskriteriums differenziert und in drei grundlegende Klassen eingeteilt [KDP+09, S. 2 f.]. Vergleiche können auf Basis (1) eines persistenten Identifikators (z. B. [AlPo03], [VPF+06]), (2) der Modellsignatur (z. B. [ReFr05]) oder (3) einem Ähnlichkeitswert zwischen Elementen der verglichenen Modelle (z. B. [TBW+07], [BrPi08], [XiSt05]) durchgeführt werden. Darüber hinaus lassen sich Vergleichsverfahren dahingehend einteilen, ob sie für eine spezifische Sprache (meist UML-Diagramme) (z. B. [XiSt05], [AlPo03]) oder sprachunabhängig (z. B. [BrPi08], [TBW+07], [VPF+06]) entwickelt wurden.

Einen Überblick über gängige Verfahren zur Berechnung von Modelldifferenzen geben KOLOVOS ET AL [KDP+09]. Als Beispiele für konkrete Implementierungen von Modellvergleichsalgorithmen seien an dieser Stelle lediglich UMFDiff [XiSt05], einem heuristischen Verfahren zur Differenzberechnung von objektorientierten Designstrukturen, SiDiff (z. B. [Schm07], [TBW+07]), einem metamodellunabhängigen und ähnlichkeitsbasierten Vergleichsalgorithmus, und der Algorithmus des EMF Compare Projekts, zur Durchführung ähnlichkeitsbasierter Vergleiche zwischen Ecore-Modellen (z. B. [BrPi08], [Toul06]), genannt.

6.4.2.2.2 Restriktionen der Differenzberechnung im SOM-R-Modellvergleichsansatz

Nachfolgend wird das Konzept für die Ermittlung von Modelldifferenzen mit dem SOM-R-Modellvergleichsansatz erläutert. Die Ausführungen abstrahieren dabei von einer algorithmischen Realisierung der Differenzberechnung, welche die Wahl eines konkreten Vergleichsalgorithmus sowie eines Modellierungswerkzeugs voraussetzen würde. Zur Gewährleistung der Plattformunabhängigkeit des SOM-R-Modellvergleichs werden stattdessen *Restriktionen an die Durchführung der Differenzberechnung* auf den Ebenen des SOM-R-Architekturmodells bestimmt:

- *Ziel des SOM-R-Modellvergleichs* ist die Bestimmung der Unterschiede zwischen zwei Versionen eines Schemas auf den Modellebenen des SOM-R-Architekturmodells. Den eingesetzten Modellierungssprachen liegt ein gemeinsames Meta-Metamodell zugrunde.
- *Erlaubte Operationen* zur Durchführung von Modelländerungen sind „Hinzufügen“ und „Entfernen“ von Modellbausteinen und der Beziehungen zwischen den Bausteinen. Die Manipulation von Eigenschaften der Modellbausteine wird nicht als eigene Operation betrachtet.
- Das *Vergleichsverfahren* berechnet die Gleichheit oder Ungleichheit zwischen Modellbausteinen zweier Schemaversionen. Die Ermittlung der Ähnlichkeit zwischen Bausteinen wird nicht untersucht.
- Die Bestimmung von Modelldifferenzen erfolgt auf Basis des *persistenten Identifikators* der Bausteine (*Art des Vergleichsverfahrens*).
  In der SOM-R-Architektur ist jedem Modellbaustein ein eindeutiger Bezeichner nach dem Schema *Metaobjekt:Objektname[.Version]* zugeordnet (vgl. Abschnitt 6.3.1). Dieser wird auch zur Spezifikation von Beziehungen verwendet.
- Das *Ergebnis der Differenzberechnung* zwischen einem $Modell_N$ und dem $Modell_{N+1}$ als dessen neue Version (Input) sind die Mengen der gemeinsamen, und somit gleichen, Bausteine (*Korrespondenzen*) sowie der eingefügten und gelöschten Bausteine und ihrer Beziehungen (*Modell-Delta*$_{N, N+1}$).

Gemäß den vorgestellten Ergebnissen einer Differenzberechnung lassen sich drei logische Schritte zur Durchführung des Modellvergleichs in der SOM-R-Methodik bestimmen:

1. Für jeden Baustein in $Modell_{N+1}$ ist zu prüfen, ob dieser mit einem Baustein in $Modell_N$ übereinstimmt (Vergleichskriterium: Identifikator). Die hierbei ermittelten Korrespondenzen bilden die Elemente der Menge unveränderter Modellbausteine.
2. Für jeden Modellbaustein in $Modell_{N+1}$ ist zu prüfen, ob er in $Modell_N$ *nicht* existiert. Die ermittelten Unterschiede beschreiben die Menge hinzugefügter Bausteine (+).
3. Für jeden Modellbaustein in $Modell_N$ ist zu prüfen, ob er in $Modell_{N+1}$ *nicht* existiert. Die ermittelten Unterschiede beschreiben die Menge entfernter Bausteine (–).

Die Differenzmengen von Schritt 2 und 3 spezifizieren das Modell-Delta$_{N, N+1}$ der beiden verglichenen Modellversionen (N, N+1). Die angegebenen Reihenfolgenummern der Schritte dienen ihrer Referenzierung und stellen keine Anforderung an die Realisierung eines Vergleichsalgorithmus dar.
Der beschriebene schrittweise Ablauf einer Ermittlung von übereinstimmenden, hinzugefügten und entfernten Bausteinen in zwei Modellen ist damit äquivalent zum schematischen Vorgehen in gängigen Vergleichsalgorithmen (z. B. [BrPi08, S. 18 ff.], [Toul06 S. 2 f.], [XiSt05, S. 58 f.])

6.4.2.3 Spezifikation und Repräsentation der Modelldifferenz

Die Ergebnisse der Differenzberechnung sind derart zu spezifizieren und repräsentieren[60], dass sie für die weitere maschinelle oder personelle Verarbeitung nutzbar sind [BrPi08, S. 30 f.]. Als methodischen Rahmen für die Spezifikation von Modelldifferenzen schlagen CICCHETTI ET AL [CDP07] eine metamodellunabhängiges Vorgehen vor. Der Lösungsansatz sieht hierbei eine automatisierte Ableitung von Differenz-Metamodellen aus dem Metamodell der zu vergleichenden Schemata vor [CDP07, S. 170]. Das Konzept des Differenz-Metamodells wird im Folgenden auf die Problemstellung des SOM-R-Architekturmodells angewendet und ein Metamodell zur *modellbasierten Spezifikation* von Modelldifferenzen gestaltet (siehe Abbildung 6.10). Die Einordnung der Bestandteile *Modell-Delta* und *Differenz-Metamodell* in die Metaebenen zeigt Abbildung 6.14.

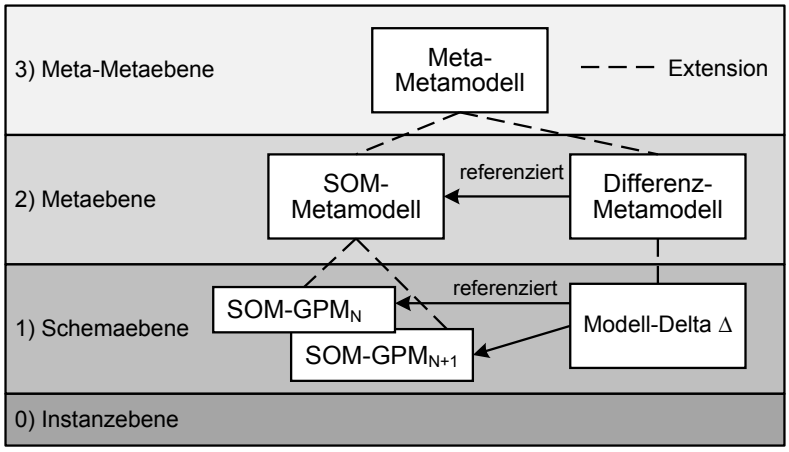


Abbildung 6.14: Metaebenen des Vergleichs von SOM-Geschäftsprozessmodellen

Das Differenz-Metamodell (Metaebene 2) definiert die Bestandteile eines Modell-Deltas (Metaebene 1). Das **Modell-Delta** besteht aus der Menge der geänderten Modellobjekte eines Schemas$_{N+1}$ im Vergleich zu einem (ursprünglichen) Schema$_N$ auf einer Modellebene des SOM-R-Architekturmodells. Die Objekte des Modell-Deltas referenzieren stets Modellobjekte in den Schemata des Systementwicklungsprozesses. Abbildung 6.15 stellt den konzeptuellen Zusam-

[60] Der Begriff der (Modell-)*Repräsentation* wird in der einschlägigen Literatur unterschiedlich verstanden (z. B. [SVE+07], [CDP07], [BrPi08], [FeSi13]). In der vorliegenden Arbeit beschreiben Metamodelle die *abstrakte Syntax* zur Spezifikation von Modellen. Als Repräsentation werden die *konkrete Syntax* und damit die Darstellung des gebildeten Modells festgelegt (z. B. [SVE+07, S. 29 f.], [FeSi13, S. 143]). Im Gegensatz dazu kann unter *Repräsentation* auch die abstrakte Syntax von Modellen verstanden werden, für die eine Menge von *Visualisierungen* (konkrete Syntax) definierbar sind [BrPi08, S. 30 ff.].

menhang zwischen Modell-Delta und Modellen beispielhaft anhand des Vergleichs von zwei Versionen eines SOM-Geschäftsprozessmodells dar.

Das *Differenz-Metamodell* der SOM-R-Methodik zeigt Abbildung 6.15. Die *Modellobjekte* auf den Modellebenen der Architektur sind die Bausteine des Modell-Deltas bzw. der Modelldifferenz. Jedem geänderten Objekt einer Schemaversion wird eine *Änderungsmarkierung* zugeordnet, die entweder das *Hinzufügen* (+) oder das *Entfernen* (-) des jeweiligen Objekts in Bezug auf die zweite Schemaversion des Vergleichs beschreibt. Jedes markierte (geänderte) Objekt steht mit beliebig vielen markierten oder nicht markierten Objekten in Beziehung. Die Beziehungen von geänderten Objekten erben die Änderungsmarkierung dieser Objekte. Beziehungen, die einen entfernten und einen hinzugefügten Baustein verknüpfen, werden als entfernt markiert. Die Entfernungsmarkierung ist demnach dominant.

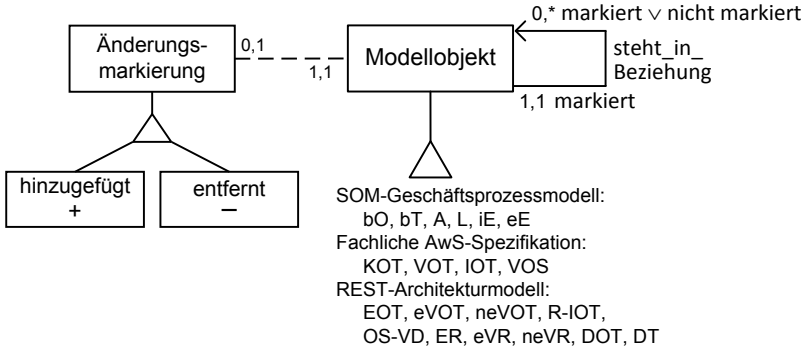


Abbildung 6.15: Differenz-Metamodell zur Spezifikation des Modell-Deltas

Die Spezifikation von (Beziehungen zu) unveränderten Elementen ist demnach auch Teil des Ergebnisses der Differenzermittlung. Diese Elemente repräsentieren gemeinsame Modellobjekte in den beiden verglichenen Schemata, die mit geänderten Bausteinen in Beziehung stehen. Die Modellobjekte sind damit Teil des instabilen Bereichs eines Schemas (siehe Abbildung 6.12).

Die Bestandteile des Modell-Delta$s_{N,\ N+1}$ sind somit als Folge von Änderungsoperationen interpretierbar, die die Weiterentwicklung von Schema$_N$ in Schema$_{N+1}$ beschreiben. Durch Umkehrung der Änderungsmarkierungen ist eine Transformation zur Kompensation der durchgeführten Schemaänderungen erzeugbar, auf deren Basis das ursprüngliche Schema$_N$ wiederhergestellt werden kann (z. B. Transformationsspezifikation in [Kelt07], [Guld09]).
Das Konzept der Protokollierung von Änderungen in einem *Changelog* wird in vielen Bereichen des Software Engineerings eingesetzt. Stellvertretend für die Vielzahl an existierenden Ansätzen sei an dieser Stelle auf die Disziplinen des Änderungs- und Versionsmanagements in Softwareentwicklungsprojekten (z. B. [Somm12, S. 743 ff.]) sowie konkrete Implementierungen wie Versionsverwaltungssysteme für Quelltexte (z. B. git und Apache Subversion[61]), Changelogs in Datenbanksystemen (z. B. [KeEi13, S. 309]) oder ChangeSummaries in Java-SDOs [GoPr08, S. 28 ff.] verwiesen. Das Modell-Delta stellt in dieser Arbeit demnach einen

[61] Webseiten: http://git-scm.com/ und http://subversion.apache.org/ (Abruf am 2. März 2015)

Lösungsvorschlag zur Implementierung von Änderungsprotokollen im Kontext der Weiterentwicklung von Modellsystemen mit SOM-R-Methodik dar.

In Zusammenhang mit der Spezifikation von Modelldifferenzen ist auch die Form ihrer Repräsentation zu bestimmen. Diese legt fest, wie ein Modell-Delta dem Nutzer präsentiert wird [BrPi08, S. 30 f.]. Als Alternativen für eine menschen-lesbare Visualisierung von Informationen bieten sich typischerweise textuelle, tabellarische oder grafische Formen an. Bei einer grafischen Präsentation des Modell-Deltas können die definierten Baustein-Symbole der zugrundeliegenden Metamodelle[62] verwendet werden. Die Änderungsmarkierung wäre durch die Erweiterung um die Symbole (+) und (–) entsprechend den Vorgaben im Differenz-Metamodell visualisierbar.

Da das Modell-Delta lediglich aus einer Menge von „losen" Bausteinen besteht und kein zusammenhängendes Modell beschrieben wird, würde eine grafische Form für menschliche Nutzer eine geringere Übersichtlichkeit bieten, als beispielsweise die textuelle Darstellung der identifizierten Modelländerungen. Eine einfache textuelle Auflistung zur Darstellung des Modell-Deltas erscheint aus diesem Grund zur Visualisierung der Beispiele dieser Arbeit vorteilhaft. Die eindeutige Spezifikation einer Änderung erfolgt durch die Angabe der *Änderungsmarkierung* (+ /–) und des *Modellobjekt-Identifikators*.

Die Anwendung des Modellvergleichsansatzes zeigt Abbildung 6.16 am Beispiel zweier Versionen des einfachen IAS eines Handelsbetriebs (Input). Der Output der Differenzberechnung sind die beiden hinzugefügten Transaktionen *Produktkatalog* und *Bestätigung* sowie ihre Beziehungen. Als Ergebnis der Modelldifferenzermittlung wird das Modell-Delta mit zwei Änderungseinträgen spezifiziert. Die Visualisierung der Ergebnisse erfolgt in Form einer textuellen Auflistung.

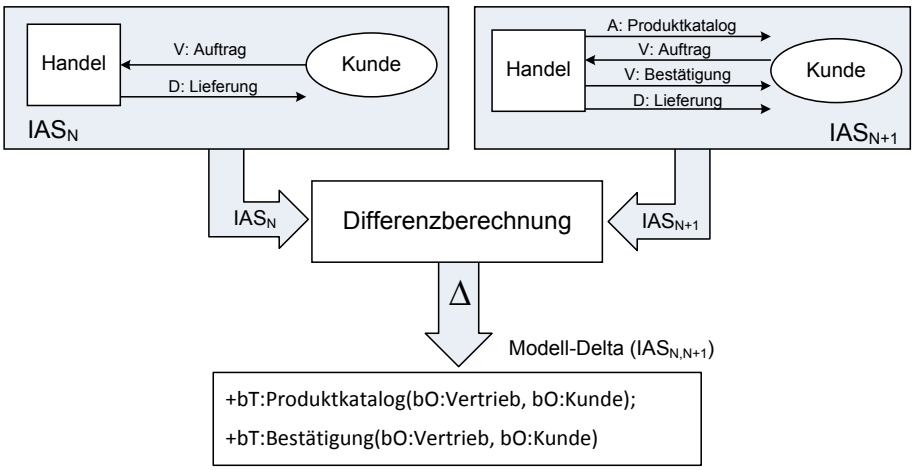


Abbildung 6.16: Textuelle Darstellung eines Modell-Deltas

Jeder Eintrag beschreibt das geänderte Modellobjekt und dessen Beziehungen auf Basis der eindeutigen Objektidentifikatoren und Angabe der Änderungsmarkierung. Beispielsweise

[62] In den Metamodellen des SOM-R-Architekturmodells wird stets eine konkrete Visualisierung von Modellobjekten durch Angabe grafischer Symbole definiert.

modelliert *+bT:Produktkatalog(bO:Vertrieb, bO:Kunde)* die Änderung *Hinzufügen* der betrieblichen Transaktion *Produktkatalog*, welche mit den betrieblichen Objekten *Vertrieb* und *Kunde* in Beziehung steht. Zur Spezifikation des Modell-Deltas wurde hier eine einfache textuelle Form gewählt, die geänderte Modellobjekte und ihre Beziehungen gemeinsam beschreibt. Eine getrennte Darstellung von Baustein und Beziehung (Einträge) zeigt Tabelle 6.7. Im weiteren Verlauf der Arbeit wird das Modell-Delta jedoch in der integrierten (einfachen) Form dargestellt.

Modellobjekt	Beziehung
+ bT:Produktkatalog	+ connects (bO:Vertrieb, bT:Produktkatalog)
	+ connects (bT:Produktkatalog, bO:Kunde)
+ bT:Bestätigung	+ connects (bO:Vertrieb, bT:Bestätigung)
	+ connects (bT:Bestätigung, bO:Kunde)

Tabelle 6.7: Tabellarische Darstellung eines Modell-Deltas

Die vorausgegangenen Ausführungen abstrahierten von einer konkreten Realisierung des SOM-R-Modellvergleichsansatzes. Die prototypische Implementierung wird in Kapitel 7 am Beispiel des Eclipse Modeling Frameworks vorgestellt. Der konzipierte Modellvergleichsansatz bildet hierfür den methodischen Rahmen. Eine Anwendung des eingeführten Ansatzes anhand der Fallstudie *Flugbuchung* und dessen integrierter Einsatz im Rahmen der Durchführung der Systemweiterentwicklung wird in Abschnitt 6.6.5 erläutert.

6.5 Konzeption eines Empfehlungsansatzes

Mit dem *Empfehlungsansatz* der SOM-R-Methodik wird in der vorliegenden Arbeit das dritte Hilfsmittel zur Unterstützung der Systemweiterentwicklung konzipiert. Der Schwerpunkt der nachfolgenden Untersuchung liegt dabei auf der Gestaltung eines grundlegenden Vorgehens zur Empfehlungsermittlung und der Spezifikation einer initialen Empfehlungsbasis. Die *Empfehlungsbasis* ist die zentrale Komponente des Ansatzes, welche *Standard-Maßnahmen* für die Behandlung der verschiedenen *Typen von Änderungen* in den Schemata von Anwendungsmodell, REST-Architekturmodell und Implementierungsmodell spezifiziert. Die Plausibilität des Konzepts und die Anwendung des Empfehlungsansatzes werden im Rahmen der Fallstudie gezeigt (Abschnitt 6.6.5).

Grundsätzlich dient die Ermittlung und Bereitstellung von Empfehlungen der Unterstützung der personellen Durchführung von Aufgaben. In der Praxis werden Empfehlungen meist maschinell durch Empfehlungssysteme (Synonym: Recommender-Systeme) generiert (z. B. [WöSc13], [TeHi01]). Das ursprüngliche Ziel eines Einsatzes von Empfehlungssystemen war die Idee, Kaufempfehlung zu generieren, beispielsweise für Bücher oder Dienstleistungen, um diese einem Nutzer, z. B. dem Besucher eines Onlineshops, vorzuschlagen [WöSc13]. Darüber hinaus hat sich die Nutzung von Empfehlungssystemen in einer Vielzahl weiterer Domänen etabliert (z. B. [KRR+10], [JFJ+09]). In diesen Domänen werden die Empfehlungen meist anhand des aktuellen Kontexts der Systemnutzung (z. B. Profil des aktuellen Nutzers, Profile ähnlicher Nutzer und Hintergrundinformationen) nutzerspezifisch gefiltert und dann bereitgestellt ([WöSc13], [JFJ+09]).

Durch die Ermittlung von Empfehlungen wird in der SOM-R-Methodik das Ziel verfolgt, Personen (Systementwickler) zu unterstützen, die an der zielgerichteten Anpassung eines geänderten Schemas im Modellsystem und der Bewältigung der bei der Durchführung dieser Aufgabe bestehenden Komplexität beteiligt sind (siehe auch Überarbeitungsschema in Abschnitt 6.6.2). Als Empfehlungen sollen dazu Hinweise auf potenzielle Auswirkungen einer Änderung sowie Handlungsempfehlungen zur Anpassung eines Schemas bereitgestellt werden. Die Informationsquellen des Ansatzes sind auf die methodischen Rahmenbedingungen (z. B. Metamodelle) sowie das vorliegende Modellsystem beschränkt. Nutzerspezifische Informationen werden in dieser Arbeit somit nicht berücksichtigt.

6.5.1 Konzeptueller Aufbau des Empfehlungsansatzes

Der konzeptuelle Aufbau des Empfehlungsansatzes sowie das Vorgehen zur Empfehlungsermittlung ist in Abbildung 6.17 dargestellt. Die Darstellung ist unabhängig von einer konkreten Modellarchitektur.

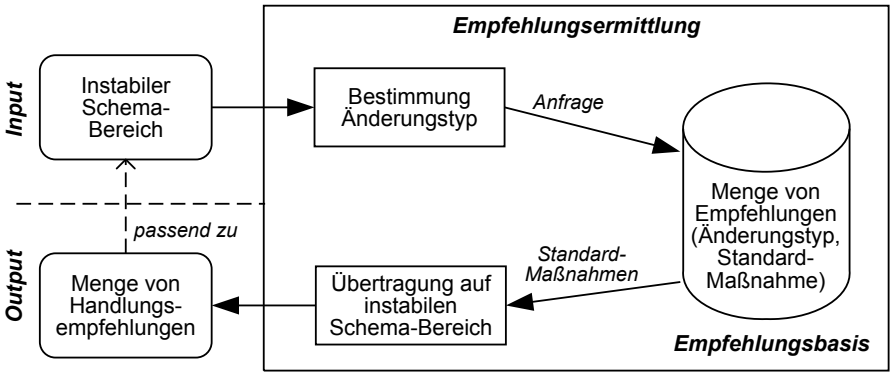


Abbildung 6.17: Bestandteile des Empfehlungsansatzes

Den Startpunkt des *Empfehlungsermittlungsprozesses* bilden die markierten Bausteine eines instabilen Schema-Bereichs (Markierung i, +, –). Zunächst werden die *Änderungstypen* der instabilen oder geänderten Bausteine bestimmt. Auf Grundlage der Änderungstypen wird eine Anfrage an die Empfehlungsbasis formuliert. Die *Empfehlungsbasis* nimmt dabei die Rolle eines Speichers ein, dessen Einträge die Standard-Maßnahmen für die definierten Änderungstypen im SOM-R-Architekturmodell beschreiben. Die Standard-Maßnahmen zu einem Änderungstyp werden aus dem Datenbestand abgefragt und anschließend auf das vorliegende Schema übertragen und parametrisiert. Durch die Übertragung und Parametrisierung von Standard-Maßnahmen auf ein konkretes Schema entstehen konkrete *Handlungsempfehlungen*, um den Systementwickler bei der Konsolidierung eines instabilen Schemabereichs zu unterstützen.

Die **Menge der Änderungstypen** im SOM-R-Architekturmodell ist durch die Kombination der definierten Baustein-Markierungen mit den Metaobjekten der Modellebenen bestimmt. Die Bausteine eines instabilen Schema-Bereichs können folgende *Markierungsarten* besitzen:

- *Änderungsmarkierung* (+ / –)
- *Instabilitätsmarkierung* (i)

Die Markierungsarten wurden bereits in Abschnitt 6.4.1 eingeführt und erläutert. Die Metamodelle der Modellebenen definieren die *Menge der Metaobjekte* (Abbildung 4.5, Abbildung 4.7 und Abbildung 5.3). Einen Überblick über die verfügbaren Bausteintypen geben zudem Tabelle 6.1, Tabelle 6.2 und Tabelle 6.3, die in Zusammenhang mit der Bildung eindeutiger Bezeichner erstellt wurden.

Die **Bestimmung der Standard-Maßnahmen** des Empfehlungsansatzes erfolgt auf Basis der *Konsolidierungsschritte* sowie der *Metamodelle* von Anwendungs-, REST-Architektur- und Implementierungsmodell (Abschnitte 5.5.3, 5.5.5 und 5.5.7). Die Metamodelle und Konsolidierungsschritte definieren alle Informationen, die erforderlich sind, um die potenziellen Auswirkungen einer Schemaänderung ermitteln und Vorschläge zu ihrer Behandlung ableiten zu können. Damit die Anpassung eines geänderten Schemas (pflegende Schema-Konsolidierung) weiterhin konsistent auf das Modellsystem ausgerichtet ist, müssen die Standard-Maßnahmen konform mit den Konsolidierungsschritten der jeweiligen Ebenen definiert werden.

6.5.2 Gestaltung der Empfehlungsbasis

Die Empfehlungsbasis der SOM-R-Methodik wird im folgenden Abschnitt gestaltet und ein initialer Datenbestand (Änderungstypen, Maßnahmen) für die Modellebenen *Anwendungs-* und *REST-Architekturmodell* erstellt. Anschließend wird untersucht, wie eine Anpassung der Empfehlungsbasis erfolgen kann.

6.5.2.1 Bestimmung von Standard-Maßnahmen

Das Ergebnis der Empfehlungsermittlung sind die abgeleiteten Handlungsempfehlungen für die Bausteine im instabilen Bereich eines Überarbeitungsschemas. Die Weiterentwicklung des Überarbeitungsschemas selbst ist eine kreative Tätigkeit, so dass die bereitgestellten Empfehlungen dem Systementwickler lediglich als Vorschläge zur Behandlung der identifizierten Änderungen dienen. Jedoch können die Handlungsempfehlungen niemals alle Einflüsse auf Entwurfsentscheidungen beschreiben bzw. berücksichtigen. Aus diesem Grund fokussieren die nachfolgenden Ausführungen auf die Erläuterung des grundlegenden Prinzips und der verfügbaren Informationen, nach denen die Festlegung von Standard-Maßnahmen für die einzelnen Änderungstypen einer Modellebene erfolgt.

METHODISCHE BASIS FÜR DIE BESTIMMUNG VON STANDARD-MAßNAHMEN

Die Bestimmung von Standard-Maßnahmen erfolgt auf Basis der instabilen Bereiche eines geänderten Schemas. Als Informationen fließen, erstens, die Änderungstypen mit

- *Art der Markierung* (+, –, i) und
- *Objekttyp (Metaobjekt)* eines Bausteins,

sowie, zweitens, das *integrierte Metamodell* einer Modellebene in die Konzeption mit ein.
Das Metamodell legt den Rahmen (verfügbare Bausteine und Beziehungen) für die Durchführung der Konsolidierung fest. Zudem sind die Auswirkungen von Schemaänderungen anhand der Beziehungen zwischen Metaobjekten bestimmbar. An dieser Stelle sei noch einmal darauf hingewiesen, dass die Schemata einer Modellebene als Projektionen auf ein integriertes

Metamodell gebildet werden. Somit definiert das Metamodell auch die Beziehungen zwischen Bausteinen unterschiedlicher Schemata, welche als Basis für die Bestimmung von Abhängigkeiten dienen.

Die *Konsolidierungsschritte* leiten die Modellbildung der Schemata einer Modellebene methodisch an und stellen die dritte Informationsquelle für die Erstellung der Empfehlungsbasis dar. Wie in Abschnitt 6.3.2.2 bereits ausgeführt, lassen sich die Konsolidierungsschritte bezüglich ihrer Wirkung in drei Kategorien einordnen:

- Änderung der Schemastruktur (Bausteine entfernen, zerlegen oder zusammenfassen)
- Spezifikation von Baustein-Eigenschaften
- Spezifikation von Abhängigkeiten zwischen Bausteinen

Die letzte (dritte) Kategorie beschreibt den Fall, dass Abhängigkeitsbeziehungen zwischen Bausteinen als Ergebnis einer Spezifikation ihrer Eigenschaften im Modellsystem entstehen. Als Beispiel sei die Realisierung des Lösungsverfahrens von Vorgangsobjekttypen in Form von Zugriffen auf eine Menge von konzeptuellen Objekttypen genannt (z. B. Abbildung 5.13).

Zur Erläuterung der Ausführungen in den nachfolgenden Abschnitten zeigt Tabelle 6.8 eine Empfehlungsbasis mit generischen Standard-Maßnahmen (metamodellunabhängig). Hierzu werden die Bestandteile der Empfehlungsbasis und die Syntax zur Beschreibung von Standard-Maßnahmen anhand der drei Änderungstypen vorgestellt.

Änderungstyp	**Generische Maßnahme**
(+) Metaobjekt	*Markiertes Objekt: Überarbeitung entsprechend Konsolidierungsschritt*. ⇒ Prüfung von Eigenschaften verknüpfter [Objekte] ⇒ Prüfung der Konsolidierung und Konsolidierungsbeziehungen der abhängigen [Objekte]
(–) Metaobjekt	*Markiertes Objekt: Keine Überarbeitung*. ⇒ Prüfung von Eigenschaften verknüpfter [Objekte] ⇒ Prüfung der Konsolidierung und Konsolidierungsbeziehungen der abhängigen [Objekte]
(i) Metaobjekt	*Überarbeitung der Objekt-Eigenschaften.* ⇒ Prüfung der Eigenschaften des Objekts, um das Lösungsverfahren mit den Beziehungen zu geänderten [Objekten] abzustimmen

Tabelle 6.8: Generische Maßnahmen zur Behandlung markierter Schemaobjekte

Der Eintrag einer Empfehlungsbasis besteht aus

- genau einem *Änderungstypen* (Definition: + / – / i *Schemaobjekt*),
- mindestens einer *Standard-Maßnahme* für die Behandlung des Änderungstyps, sowie
- beliebig vielen Maßnahmen zur Behandlung von *Auswirkungen des Änderungstyps*.

Für die Darstellung der Maßnahmen wird festgelegt, dass diese für die Behandlung des jeweils markierten Objekts durch *kursive Schrift* hervorgehoben werden. Die Maßnahmen zur Behandlung der potenziellen Auswirkungen einer Änderung auf das Schema (z. B. existenzielle Abhängigkeiten und Aufrufbeziehungen) sowie die zugehörigen Empfehlungen sind durch

einen Doppelpfeil (⇒) gekennzeichnet. Eine Auflistung der Modellbausteine, die im konkreten Änderungsfall bestimmbar sind, umfassen eckige Klammern [*Objekt*]. Als „verknüpft" werden alle instabilen Objekte bezeichnet, die mit einem geänderten Modellbaustein (betrachteter Änderungstyp) in Beziehung stehen. Die Menge der „abhängigen" [Objekte] umfasst alle direkt oder transitiv verknüpften instabilen Bausteine, einschließlich des jeweils ersten unveränderten (nicht instabilen) Bausteins des geänderten Schemas. Durch die Konkretisierung der Maßnahmen (und der definierten Platzhalter) für einen bestimmten Sachverhalt entstehen Handlungsempfehlungen.

Die Einträge der Empfehlungsbasis werden im Folgenden erläutert.
Die neu hinzugefügten (+) Objekte eines Schemas sind grundsätzlich nach den Regeln des dazu korrespondierenden Konsolidierungsschrittes zu überarbeiten und in das bestehende Schema zu integrieren. Dieses Vorgehen kann zu strukturellen Änderungen am Objekt selbst, sowie weiteren Objekten im Schema führen. Die Auswirkungen einer durchgeführten (pflegenden) Konsolidierung auf die Schemastruktur lassen sich anhand der angewandten Strukturänderungen bezüglich des Umfangs der vorgenommenen Anpassungen grob kategorisieren:

- Wird das (neu hinzugefügte) Objekt wieder *entfernt*, so resultieren aus der Änderung *Hinzufügen* keine weiteren Auswirkungen auf das Schema.
- Wird das (+) Objekt *konsolidiert*, jedoch *nicht* mit weiteren Objekten *zusammengefasst*, so sind die Auswirkungen auf das Schema gering, da lediglich die geänderten Beziehungen zu den verknüpften Objekten des (+)Objekts zu prüfen und ggf. zu spezifizieren sind.
- Wird das (+) Objekt *konsolidiert* und mit weiteren Objekten *zusammengefasst*, so sind größere Auswirkungen auf das Schema zu erwarten, da der instabile Schemabereich, für den die Auswirkungen der strukturellen Überarbeitungen und Entwurfsentscheidungen zu prüfen sind, wächst.

Im Falle des *Entfernens* eines Objekts ist für das markierte Objekt selbst keine Überarbeitung erforderlich. Die Maßnahmen beschränken sich auf die Prüfung und Behandlung von Auswirkungen der potenziell abhängigen Objekte sowie der zugehörigen Konsolidierungsbeziehungen, wenn diese z. B. durch angepasste Entwurfsentscheidungen aufgrund des Entfernens entstanden sind.

Die Bausteine mit *Instabilitätsmarkierung* sind, sofern nicht von Auswirkungen anderer Maßnahmen betroffen, bezüglich der Realisierung von Verknüpfungen zu geänderten Objekten anzupassen. Weitere Auswirkungen auf abhängige Objekte bestehen normalweise nicht, sind jedoch zu prüfen.

Die vorgestellten generischen Maßnahmen werden in den Abschnitten 6.5.2.2 bis 6.5.2.4 im Hinblick auf die Bausteine, die Baustein-Beziehungen sowie die zugehörigen Schritte der Konsolidierung für die Modellebenen der SOM-R-Architektur konkretisiert. Hierzu werden die Standard-Maßnahmen von Änderungstypen auf der Ebene des Anwendungsmodells und REST-Architekturmodells definiert und erläutert. Die Spezifikation der potenziellen Auswirkungen umfasst dabei lediglich Empfehlungen für typische Abhängigkeiten der Modellebene, aber

nicht alle Metamodell-Beziehungen. Die Definition von Standard-Maßnahmen für das Implementierungsmodell setzt die Wahl konkreter Technologien und die Gestaltung der plattformspezifischen Zielarchitektur voraus. Sie erfolgt deshalb nur in generischer Form.

6.5.2.2 Standard-Maßnahmen für die fachliche AwS-Spezifikation

Der instabile Bereich von KOS und VOS (Anwendungsmodell) besteht aus Instanzen der Metaobjekte *Objekttyp* und *Beziehung* zwischen zwei Objekttypen. Abbildung 6.18 visualisiert die Bausteine der Modellebene *fachliche AwS-Spezifikation* und ihre Beziehungen untereinander in Form einer Projektion auf das SOM-Metamodell (Abbildung 4.7).

Die (Meta-)Beziehung zwischen den Metaobjekten ermöglicht die Bestimmung der Auswirkungen von Objektänderungen in einem konkreten Schema auf Basis der definierten Abhängigkeiten des Metamodells. Die Aufrufbeziehungen zwischen Objekttypen unterschiedlicher Schemata sind ebenfalls explizit dargestellt, um die Analyse der Auswirkungen einer Änderung zu unterstützen.

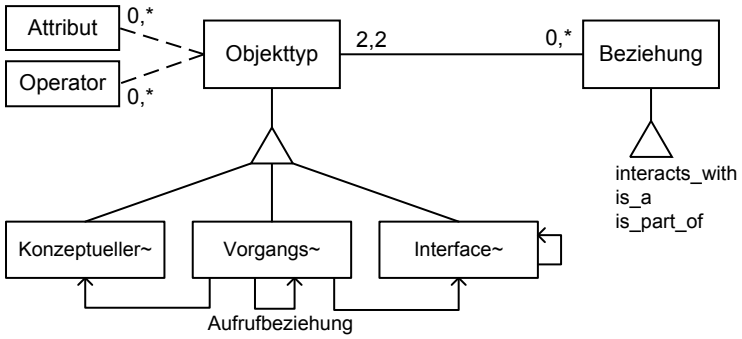


Abbildung 6.18: Metaobjekte und ihre Beziehungen im Anwendungsmodell (vereinfachte Darstellung)

6.5.2.2.1 Änderungstyp Hinzufügen (+)

Die Standard-Maßnahmen zur Behandlung des Änderungstyps „Hinzufügen (+)“ auf der Ebene des Anwendungsmodells fasst Tabelle 6.9 zusammen. Die änderungstypspezifischen Auswirkungen (⇒) werden den jeweiligen Typen direkt zugeordnet. Dagegen bilden Auswirkungen, die für eine Menge von Änderungstypen gelten einen eigenen Eintrag. So sind die Maßnahmen des Änderungstyps *+ Objekttyp* (neu hinzugefügter Objekttyp) für + KOT und + VOT[63] gültig.

Die Behandlung von Generalisierungen (is_a-Beziehung) und Aggregationen (is_part_of-Beziehung) zwischen einem instabilen und einem hinzugefügten Objekttypen sind nicht zu berücksichtigen, da diese durch Zerlegung komplexer Objekttypen entstehen. Sie treten im initialen KOS somit nicht in Form hinzugefügter Schemaobjekte auf.

[63] Änderungstypen werden in den Standard-Maßnahmen über ihre Abkürzung referenziert (z. B. + KOT). In der Spalte der empfohlenen Standard-Maßnahmen werden die Bausteine eines Schemas und ihre Markierung ebenfalls in abgekürzter Form referenziert. Beispielsweise bezieht sich „ (i) [Objekttypen]“ auf die „Menge der instabil markierten Objekttypen“ eines Schemas, die mit der Instanz des Änderungstyps verknüpft sind.

Änderungstyp	Standard-Maßnahme
+ KOT (OOT, LOT, TOT)	*Überarbeitung nach Konsolidierung K1.1*
	⇒ Prüfen der Anpassung von VOT-Lösungsverfahren im [VOS] hinsichtlich des + KOT
+ VOT	*Überarbeitung nach Konsolidierung K1.2*
	⇒ Prüfen der IOT-Zuordnung und -Konsolidierung ⇒ Prüfen der VOT-Lösungsverfahren bezüglich + VOT
+ Objekttyp	⇒ Prüfen von Eigenschaften verknüpfter (i) [Objekttypen] ⇒ Prüfen der Konsolidierung abhängiger [Objekttypen]
+ interacts_with–Beziehung	*Überarbeitung nach Konsolidierung K1.1 (KOS) und/oder K1.2 (VOS)*
	⇒ Prüfen von Lösungsverfahren der [Objekttypen], die durch + interacts_with-Beziehung verknüpft sind (Empfang/Senden von Nachrichten)

Tabelle 6.9: Standard-Maßnahmen (fachliche AwS-Spezifikation) – Änderungstyp *Hinzufügen*

6.5.2.2.2 Änderungstyp Entfernen (–)

Die Standard-Maßnahmen zur Behandlung des Änderungstyps „Entfernen (–)“ fasst Tabelle 6.10 für das Anwendungsmodell zusammen.

Änderungstyp	Standard-Maßnahme
– Objekttyp	*Keine Überarbeitung*
	⇒ Prüfen von Eigenschaften verknüpfter (i) [Objekttypen] ⇒ Prüfen der Konsolidierung abhängiger [Objekttypen]
– KOT (OOT, LOT, TOT)	⇒ Prüfen der Anpassung von VOT-Lösungsverfahren im [VOS] hinsichtlich des – KOT
– VOT	⇒ Prüfen von VOT-Lösungsverfahren bezüglich – VOT
– IOT	⇒ Prüfen von VOT-Lösungsverfahren bezüglich – IOT
– interacts_with-Beziehung	*Keine Überarbeitung*
	⇒ Prüfen von Eigenschaften der [Objekttypen], die durch die – Interaktionsbeziehung verknüpft waren (Empfang/Senden von Nachrichten)

Tabelle 6.10: Standard-Maßnahmen (fachliche AwS-Spezifikation) – Änderungstyp *Entfernen*

Die Lösungsverfahren von VOTs, denen entfernte KOTs als Attribute zugeordnet waren, sind zu prüfen und bei Bedarf anzupassen. Falls entfernte interacts_with-Beziehungen existierende Objekttypen verbinden, so sind die Lösungsverfahren der Objekttypen mit dem Entfernen von Beziehungen abzustimmen. Zum Beispiel sind Operationen von Objekttypen anzupassen, die den Empfang oder das Senden von Nachrichten zu einem entfernten (verknüpften) Objekttypen realisieren.

6.5.2.2.3 Änderungstyp Instabilität (i)

Die Standard-Maßnahmen zur Behandlung von Objekten mit einer „Instabilitätsmarkierung (i)“ fasst Tabelle 6.11 für das Anwendungsmodell zusammen.

Änderungstyp	Standard-Maßnahme
(i) Objekttyp	*Anpassen von Eigenschaften* zur Spezifikation der Interaktion (Empfang/ Senden von +/-Nachrichten einer Aufrufbeziehung) mit geänderten [Objekttyp]
	⇒ Prüfen verknüpfter (i) [Objekttypen]

Tabelle 6.11: Standard-Maßnahmen (fachliche AwS-Spezifikation) – Änderungstyp *Instabilität*

Die Anpassung der Eigenschaften eines bestehenden Bausteins wird durch das Hinzufügen einer neuen Version des Bausteins realisiert. Der Fall einer instabil markierten interacts_with-Beziehung ist nicht dargestellt, da diese existenzabhängig von den verbundenen Objekttypen sind. Demnach kann eine unveränderte interacts_with-Beziehung nicht mit einem als geändert markierten Objekttypen in Beziehung stehen.

6.5.2.3 Standard-Maßnahmen für das REST-Architekturmodell

Objekttypen, Beziehungen und Ressourcen sowie Datenobjekt- und Dokumenttypen bilden die zentralen Modellbausteine des REST-Architekturmodells (vgl. Abbildung 5.6). Objekttypen werden als Entitätsobjekttypen, (elementarer/nicht-elementarer) Vorgangsobjekttypen oder Interfaceobjekttypen (REST-Ebene) spezialisiert. Die Funktionalität von Objekttypen wird in Form von Entitäts- und Vorgangsressourcen veröffentlicht.

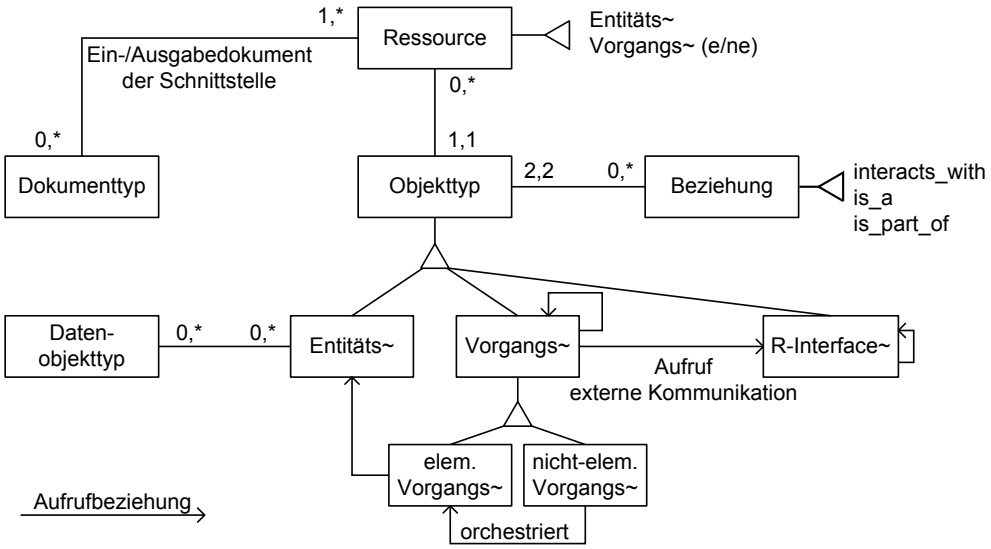


Abbildung 6.19: Metaobjekte und ihre Beziehungen im REST-Architekturmodell (vereinfachte Darstellung)

Die Kernbausteine des REST-Metamodells und ihre Abhängigkeitsbeziehungen zeigt Abbildung 6.19. Die Aufrufbeziehungen zwischen Objekttypen werden zusätzlich hervorgehoben. Auf Basis dieser Informationen werden die Änderungstypen des REST-Architekturmodells und ihre Auswirkungen bestimmt. Die Spezifikation von Eigenschaften der REST-Bausteine erfolgt in der Konsolidierung K2. Zur besseren Übersichtlichkeit wird deshalb auf ihre Darstellung verzichtet.

6.5.2.3.1 Änderungstyp Hinzufügen (+)

Die Standard-Maßnahmen zur Behandlung des Änderungstyps „Hinzufügen (+)“ fasst Tabelle 6.12 für die Ebene des REST-Architekturmodells zusammen. Die Abhängigkeiten zwischen neu hinzugefügten Bausteinen definiert Transformationsschritt T2. Beispielsweise resultiert ein neuer KOT im konsolidierten KOS stets mit einem neu hinzufügten EOT, seinen Entitätsressourcen sowie Datenobjekt- und Dokumenttypen.

Änderungstyp	Standard-Maßnahme
+ EOT	*Überarbeitung nach Konsolidierung K2.1*
	⇒ Prüfen verknüpfter (i) [EOT] zur Integration des + EOT in das OS-ED (+ Eigenschaften zur Realisierung des IN/OUT von Nachrichten) ⇒ Konsolidieren abhängiger [ER], [DT] und [DOT] ⇒ Prüfen der Lösungsverfahren abhängiger [VOT]
+ eVOT	*Überarbeitung nach Konsolidierung K2.2.1*
	⇒ Prüfen verknüpfter (i) [VOT] zur Integration von + eVOT in [OS-VD] (+ Eigenschaften zur Realisierung des IN/OUT von Nachrichten) ⇒ Konsolidieren abhängiger [VR] und [DT] (und [neVOT] bei Orchestrierung) ⇒ Prüfen der Zuordnung und Konsolidierung von R-IOTs
+ IOT-R	*Überarbeitung nach Konsolidierung K2.2.2*
+ Beziehung (OT_X, OT_Y)	*Überarbeitung nach Konsolidierung K2.2.1 oder K2.2.2* (interacts_with)
	⇒ Prüfen von Eigenschaften der zwei verknüpften Objekttypen OT_X und OT_Y (+ Eigenschaften zur Realisierung des IN/OUT von Nachrichten)
+ Ressource (ER, VR)	*Überarbeitung nach Konsolidierung nach K2.1 (ER) oder K2.2.1 (VR)*
	⇒ Prüfen verknüpfter (i) [Objekttypen], die auf die + Ressource zugreifen (+ Eigenschaften zur Realisierung des IN/OUT von Nachrichten)
+ DT	*Überarbeitung nach Konsolidierung K2.3*
	⇒ Prüfen von Eigenschaften (i) der verknüpften [Ressourcen] und [Objekttypen] (Realisierung des IN/OUT von + Nachricht)
+ DOT	*Überarbeitung nach Konsolidierung K2.4*
	⇒ Prüfen der Attribute verknüpfter (i) [DOT]
+ Beziehung (DOT_X, DOT_Y)	*Überarbeitung nach Konsolidierung K2.4*
	⇒ Prüfen der Schlüsselreferenz verknüpfter (i) [DOT]

Tabelle 6.12: Standard-Maßnahmen (REST-Architekturmodell) – Änderungstyp *Hinzufügen*

Die nicht-elementaren VOTs und ihre Ressourcen werden aus eVOTs nach den Regeln von Konsolidierung K2.2.1 gebildet. Für neVOTs würde somit die Maßnahme „Überarbeitung nach Konsolidierung K2.2.1“ gelten. Als Maßnahmen zur Behandlung der weiteren Auswirkungen eines + neVOT sind eine „⇒ Prüfung verknüpfter (i) neVOTs (+ IN/OUT-Nachrichten benachbarter neVOTs)“ und die „⇒ Konsolidierung abhängiger neVR und DT“ zu spezifizieren.

6.5.2.3.2 Änderungstyp Entfernen (–)

Tabelle 6.13 fasst die Standard-Maßnahmen zur Behandlung der Änderungstypen von „(–) Entfernen“ auf der Modellebene des REST-Architekturmodells zusammen. Die entfernten Schemaobjekte selbst werden nicht überarbeitet. Die vorgeschlagenen Maßnahmen zielen auf die Kompensation des Wegfalls der Funktionalität von entfernten Bausteinen ab.

Änderungstyp	Standard-Maßnahme
– Objekttyp	*Keine Überarbeitung.*
– EOT	⇒ Prüfen verknüpfter (i) [EOT] zur Berücksichtigung von – EOT in OS-ED (Anpassen der Eigenschaften zur Berücksichtigung des Wegfalls von IN/OUT-Nachricht) ⇒ Prüfen des Entfernens (/ Anpassens) abhängiger [ER], [DT] und [DOT] ⇒ Prüfen von Lösungsverfahren abhängiger [VOT]
– eVOT	⇒ Prüfen verknüpfter (i) [VOT] zur Berücksichtigung von – eVOT in [OS-VD] (Anpassen der Eigenschaften zur Berücksichtigung des Wegfalls von IN/OUT-Nachricht) ⇒ Prüfen des Entfernens (/ Anpassens) abhängiger [VR], [DT] und [R-IOT]

	⇒ Prüfen der Lösungsverfahren von [neVOT] (bei Orchestrierung)
– neVOT	⇒ Prüfen von Eigenschaften verknüpfter (i) [eVOT] und Beziehungen zur Gestaltung der Service-Koordination (aufgrund entfernter Orchestrierung) ⇒ Prüfen von Eigenschaften verknüpfter (i) [neVOT] ⇒ Prüfen des Entfernens (/ Anpassens) abhängiger [neVR], [DT] und [R-IOT]
– R-IOT	⇒ Prüfen der Kommunikationsbeziehungen zu (i) [VOT] der externen Komponenten
– Beziehung (OT_X, OT_Y)	⇒ Prüfen von Eigenschaften der zwei verknüpften Objekttypen OT_X und OT_Y (Anpassen ihrer Eigenschaften zur Berücksichtigung der entfernten Beziehung / IN/OUT-Nachricht)
– Ressource (ER, VR)	⇒ Prüfen verknüpfter (i) [Objekttypen], die auf die – Ressource zugreifen (Anpassen ihrer Eigenschaften zur Berücksichtigung – IN/OUT von Nachricht)
– DT	⇒ Prüfen der *Eigenschaften* verknüpfter (i) [Ressourcen] und [Objekttypen] (Berücksichtigung der entfernten Parameter von Anfrage/Antwort-Nachricht)
– DOT	⇒ Prüfen der Attribute verknüpfter (i) [DOT]
– Beziehung (DOT_X, DOT_Y)	⇒ Prüfen der Schlüsselreferenz verknüpfter [DOT]

Tabelle 6.13: Standard-Maßnahmen (REST-Architekturmodell) – Änderungstyp *Entfernen*

Das Entfernen von Modellbausteinen, aus denen nicht-elementare VOTs abgeleitet wurden, führt zum Änderungstyp „– neVOT“. Wesentliche Auswirkung dieser Änderung auf das existierende Schema ist der Wegfall der Dienst-Orchestrierung, die durch die Wahl und Gestaltung einer alternativen Koordinationsform zu behandeln ist.

6.5.2.3.3 Änderungstyp Instabilität (i)

Tabelle 6.14 zeigt die Standard-Maßnahmen zur Behandlung des Änderungstyps der Instabilitätsmarkierung (i) von Bausteinen auf der Modellebene REST-Architekturmodell. Die Ursache der Instabilität von Ressourcen, Dokumenttypen und Datenobjekttypen liegt zumeist in einer Instabilität und einem Anpassen von abhängigen (verknüpften) Objekttypen. Ein unveränderter instabiler Modellbaustein ist grundsätzlich im Hinblick auf die Bewältigung der Änderungen seiner Umwelt zu prüfen und ggfs. zu überarbeiten.

Änderungstyp	Standard-Maßnahme
(i) Objekttyp	*Prüfen von Eigenschaften* zur Spezifikation der Interaktion (Empfang/Senden von Nachrichten) mit geänderten (+/-) [Bausteinen]
	⇒ Prüfen verknüpfter (i) [Objekttypen] ⇒ Prüfen der Eigenschaften und ggfs. Konsolidierung von abhängigen [Ressourcen], [DT] und [DOT]
(i) Ressource	*Prüfen von Eigenschaften* bzgl. der Interaktion mit geänderten (+/-) [Bausteinen]
(i) DT	*Prüfen von Attributen*
(i) DOT	*Prüfen von Attributen* zur Realisierung von Referenzen zu verknüpften [DOT]

Tabelle 6.14: Standard-Maßnahmen (REST-Architekturmodell) – Änderungstyp *Instabilität*

6.5.2.4 Maßnahmen zur Anpassung des Implementierungsmodells

Das Ziel des dritten Transformationsschrittes ist die automatisierte Erzeugung des lauffähigen Systems RESTful SOA durch die Generierung von Programmcode (Abschnitt 5.5.6). Im Falle eines vollständigen Erreichens dieser Zielsetzung müssten keine weiteren Anpassungen am

Implementierungsmodell durchgeführt werden, da alle Schemaänderungen bereits im REST-Architekturmodell im Zuge der (pflegenden) Konsolidierung behandelt worden sind.

Häufig sind auf der Implementierungsebene jedoch Ergänzungen am generierten Softwaresystem in einem separaten Konsolidierungsschritt vorzunehmen, um dieses in einen lauffähigen Zustand zu überführen. Da die grundlegende Struktur der RESTful SOA auf der Ebene des REST-Architekturmodells definiert wird, beschränken sich die Anpassungen des Implementierungsmodells i. d. R. auf die Spezifikation von nicht-generierbaren Systemeigenschaften. Typischerweise werden hier die Lösungsverfahren von Methoden spezifiziert oder die Basismaschine konfiguriert (Abschnitt 5.5.7). Tabelle 6.15 zeigt beispielhaft Maßnahmen zur Behandlung der Änderungstypen „Hinzufügen und Entfernen von Diensten sowie Relationstypen“.

Änderungstyp	Standard-Maßnahme
+ Entitätsdienste	*Plattformspezifische Realisierung der Lösungsverfahren* ⇒ Prüfen der Konfiguration der Basismaschine
– Entitätsdienste	*Keine Überarbeitung* ⇒ Ablösen / Versionieren des Dienstes im Rahmen des Änderungsmanagements ⇒ Prüfen der Konfiguration der Basismaschine
+ Relationstyp	*Keine Überarbeitung* (Schemadefinition vollständig generiert) ⇒ Prüfen und Anpassen der Datenbankabfragen verknüpfter [Entitätsdienste]
– Relationstyp	*Keine Überarbeitung* ⇒ Prüfen der Datenbankabfrage verknüpfter [Entitätsdienste]

Tabelle 6.15: Standard-Maßnahmen (Implementierung) – Beispiel der Änderung von Entitätsdiensten

6.5.3 Pflege der Empfehlungsbasis

Der im vorangegangenen Abschnitt 6.5.2 spezifizierte Datenbestand der Empfehlungsbasis ist mit einer initialen Menge an Maßnahmen zur Behandlung von Änderungstypen auf den Ebenen des SOM-R-Architekturmodells gefüllt. Mit dem fortschreitenden Einsatz der SOM-R-Methodik in Entwicklungsprojekten ist zu erwarten, dass weitere Erfahrungen über die Bewältigung von Änderungstypen gesammelt oder auch Erkenntnisse über typische Auswirkungen einer Änderung auf die implementierte RESTful SOA gewonnen werden. Um derartiges Wissen in zukünftigen Projekten berücksichtigen zu können, bietet sich die Pflege der Empfehlungsbasis an. Grundsätzlich kann der Datenbestand durch folgende Operationen gepflegt werden:

- Hinzufügen neuer Einträge
- Aktualisieren existierender Einträge
- Entfernen existierender Einträge

Jeder neue oder aktualisierte Eintrag spezifiziert einen Änderungstypen, eine Menge von Standard-Maßnahmen sowie, optional, eine Menge von Auswirkungen. Die Definition der neuen Einträge muss hierbei konform mit dem korrespondierenden Metamodell und der Konsolidierung des geänderten Metaobjekts erfolgen.

Denkbar ist somit auch das Erstellen von „komplexen“ Änderungstypen, die eine logische Verknüpfung von Änderungstypen und damit einen Teilbereich in einem instabilen Schema

definieren. Tabelle 6.16 zeigt die beispielhafte Erweiterung der Empfehlungsbasis um einen neuen Eintrag, der die Standard-Maßnahmen zur Behandlung des entfernten Teils einer Aggregationsbeziehung festlegt.

Änderungstyp	Standard-Maßnahme
– EOT_{Teil} $\wedge$ – is_part_of (EOT_{Ganzes}, EOT_{Teil})	*Keine Überarbeitung (– EOT_{Teil} und* – is_part_of-Beziehung) *Überarbeitung und Prüfung der Integration von Eigenschaften des – EOT_{Teil} (EOT_{Ganzes})* $\Rightarrow$ Prüfen verknüpfter (i) [EOT] zur Berücksichtigung von – EOT_{Teil} in OS-ED (Anpassen von Eigenschaften zur Berücksichtigung des – IN/OUT von Nachrichten) $\Rightarrow$ Prüfen des Entfernens (/ Anpassens) abhängiger [ER], [DT] und [DOT] (EOT_{Teil}) $\Rightarrow$ Prüfen des Anpassens von [ER], [DT] und [DOT] zur Berücksichtigung der Überarbeitung von EOT_{Ganzes} $\Rightarrow$ Prüfen der Lösungsverfahren abhängiger [VOT]

Tabelle 6.16: Exemplarische Erweiterung der Empfehlungsbasis

Der definierte Änderungstyp ist hierbei aus zwei elementaren Änderungstypen zusammengesetzt (– EOT $\wedge$ – is_part_of-Beziehung), die durch ein logisches UND ($\wedge$) verknüpft werden.

Die Anwendung des konzipierten SOM-R-Empfehlungsansatzes wird in der vorliegenden Arbeit im Rahmen der Weiterentwicklung des spezifizierten Modellsystems der Fallstudie demonstriert und konkret erläutert (Abschnitt 6.6.5). In der Fallstudie werden die Handlungsempfehlungen dabei auf Grundlage der initialen Empfehlungsbasis generiert. Die Pflege des definierten Datenbestandes wird nicht betrachtet.

6.6 Konzeption des Lösungsverfahrens der modellbasierten Systempflege

Der Gegenstand des vorliegenden Abschnitts ist die Konzeption des Lösungsverfahrens zur modellbasierten Pflege von Informationssystemen. Das Lösungsverfahren beschreibt die integrierte Nutzung von Dokumentations-, Modellvergleichs- und Empfehlungsansatz als Hilfsmittel der SOM-R-Methodik, die in den Abschnitten 6.3, 6.4 und 6.5 konzipiert und vorgestellt wurden. Der schematische Ablauf des Lösungsverfahrens wurde bereits in Abschnitt 6.2.3 als Vorgehensmodell der Weiterentwicklung eingeführt. Es spezifiziert die schrittweise Ausführung von Analyse- und Pflegeaktivitäten zur Anpassung eines SOM-R-Architekturmodells.

Das methodische Fundament für die Durchführung der Systemweiterentwicklungsaufgabe bilden die drei Hilfsmittel als Bestandteile der erweiterten SOM-R-Methodik:

- Der *Dokumentationsansatz* archiviert die Ergebnisse der Modellbildung durch die Speicherung der konsolidierten Schemata der Modellebenen und der strukturellen Entwurfsentscheidungen der Schema-Konsolidierung. Sie stellen die nicht-automatisiert erzeugbaren Gegenstände des Systementwicklungsprozesses dar.
- Als Startpunkt der Weiterentwicklung werden die Unterschiede zwischen zwei Schema-Versionen mit dem *Modellvergleichsansatz* ermittelt. Die Menge geänderter Modellbausteine einer Schemaversion sowie deren Beziehungen zu unveränderten oder veränderten Modellbausteinen werden in einem *Modell-Delta* spezifiziert.

- Zur Unterstützung der personellen Aufgabe bei der Anpassung von instabilen Bereichen eines Schemas werden Handlungsempfehlungen mit dem *Empfehlungsansatz* abgeleitet.

Das Anpassen des geänderten Modellsystems erfolgt auf Basis eines Überarbeitungsschemas, dessen Erzeugung im Abschnitt 6.6.3 vorgestellt wird. Die Weiterentwicklung bzw. Pflege der Ebenen des SOM-R-Modellsystems wird weiterhin auf Basis der integrierten Metamodelle (Abschnitt 5.3) sowie der Transformations- und Konsolidierungsregeln (Abschnitt 5.5) methodisch angeleitet.

6.6.1 Struktur und Ablauf des Prozesses der Systempflege

Das schematische Vorgehensmodell zur modellbasierten Pflege von SOM-R-Modellsystemen wird nachfolgend verfeinert. Die Aktivitäten und Entwicklungsschritte zur Durchführung der drei Hauptphasen Anforderungsanalyse und -definition (A), Softwareentwurf (D) und Realisierung (R) zeigt Tabelle 6.17. Sie spezifizieren das Vorgehen zur schrittweisen Anpassung und der konsistenten Ausrichtung eines SOM-R-Modellsystems auf die fachlichen Anforderungen eines veränderten SOM-Geschäftsprozessmodells$_{N+1}$. Damit wird der Prozess zur Weiterentwicklung des modellierten Informationssystems beschrieben.

Modellebenen		*GPM*	*Anwendungsmodell*				*REST-Architekturmodell*				*Implementierungsmodell*		
Entwicklungsschritte / **Hauptphasen**		Analyse Strukturflexibilität Bestimmen der Modelldifferenz	Bestimmen der Modelldifferenz	Ermitteln des Überarbeitungsschemas	Ableiten von Handlungsempfehlungen	Konsolidieren des Überarbeitungsschemas	Bestimmen der Modelldifferenz	Ermitteln des Überarbeitungsschemas	Ableiten von Handlungsempfehlungen	Konsolidieren des Überarbeitungsschemas	Bestimmen von Modelldifferenz und Überarbeitungsschema	Ableiten von Handlungsempfehlungen	Realisieren der RESTful SOA
A	*Analyse*	G.A1	A.A1	A.A2									
	Pflege				A.P1	A.P2							
D	*Analyse*						D.A1	D.A2					
	Pflege								D.P1	D.P2			
R	*Analyse*										R.A1, R.A2		
	Pflege											R.P1	R.P2
SOM-Geschäftsprozessmodell$_{N+1}$			**Anwendungsmodell$_{N+1}$**				**REST-Architekturmodell$_{N+1}$**				**Implementierungsmodell$_{N+1}$**		

Tabelle 6.17: Aktivitäten und Entwicklungsschritte des Weiterentwicklungsprozesses

Das SOM-Geschäftsprozessmodell$_{N+1}$ stellt den Startpunkt für die Ableitung der initialen Schemata des Anwendungsmodells dar und löst damit die Durchführung des Entwicklungsschrittes A.A1 aus (graue Schrift). Um die Auswirkungen von Änderungen in strukturflexiblen SOM-Geschäftsprozessmodellen auf den Ebenen des SOM-R-Modellsystems und der implementierten Zielarchitektur RESTful SOA analysieren zu können, wird das Modell-Delta auf der GP-Ebene ebenfalls bestimmt (Analyse G.A1).

Die Anpassung der Modellebenen von Anwendungsmodell und REST-Architekturmodell beginnt in Schritt A1 mit der Bestimmung der Modelldifferenz zwischen den initialen Schemata$_{N+1}$ und ihren jeweiligen (Vor-)Versionen (initialen Schemata$_N$). Anschließend wird das ermittelte Modell-Delta auf Basis der dokumentierten Konsolidierungsbeziehungen in ein *Überarbeitungsschema* überführt (Schritt A2). Das Überarbeitungsschema stellt das zentrale Ergebnis der Analyseaktivität (A1 und A2) dar und bildet den Input der ersten Pflegeaktivität der jeweiligen Hauptphase. In Schritt P1 wird die Ableitung von Handlungsempfehlungen durchgeführt. Die Pflege der konsolidierten Schemata einer Modellebene entspricht einer nicht-automatisierbaren Entscheidungsaufgabe, zu deren Durchführung das Treffen von Entwurfsentscheidungen erforderlich ist. Die ermittelten Handlungsempfehlungen, dokumentierten Konsolidierungsbeziehungen und die Modell-Deltas der initialen Schemata unterstützen den Systementwickler bei der Konsolidierung der Überarbeitungsschemata (Schritt P2). Das Ergebnis der Pflegeaktivität (P1 und P2) einer Modellebene sind die konsolidierten Schemata$_{N+1}$, welche den Input für die Weiterentwicklung der nachfolgenden Modellebene bilden.

Da das Implementierungsmodell in einem modellgetriebenen Schritt spezifiziert wird, erfolgen auf dieser Modellebene normalerweise keine strukturellen Überarbeitungen der Schemata. Es werden lediglich Anreicherungen der Systemspezifikation, z. B. mittels Einfügen von individuellem Code, oder Konfigurationen an der Basismaschine, durchgeführt. Aus diesem Grund ist die Bestimmung der Modelldifferenz und der zu überarbeitenden Bereiche des Implementierungsmodells zu einem gemeinsamen Analyseschritt zusammengefasst (R.A1, R.A2). Das Ermitteln von Handlungsempfehlungen zur Weiterentwicklung des Softwaresystems ist auf die Konkretisierung des abgeleiteten Programmcodes (R.P1) beschränkt. Ergänzend ist die Durchführung weiterer Entwicklungsaufgaben wie z. B. die Umsetzung der Service-Versionierung oder die Evolution der Datenbasis denkbar. Mit der Implementierung und Inbetriebnahme der RESTful SOA$_{N+1}$ ist die Weiterentwicklungsaufgabe abgeschlossen.

Die Modelldifferenz zweier initialer Schemata ist in der SOM-R-Methodik typischerweise auf zwei alternativen Wegen bestimmbar. Im ersten Fall erfolgt die Ermittlung des Modell-Deltas für die konsolidierten Schemata$_{N+1}$ einer Ebene (z. B. SOM-GPM$_{N+1}$), die anschließend durch Anwendung der Transformationsspezifikation in die initialen Schemata sowie die korrespondierenden Modell-Deltas der abgeleiteten (tieferen) Modellebene (z. B. initialen KOS und VOS) überführt werden. Im zweiten Fall werden zunächst die initialen Schemata$_{N+1}$ der tieferen Ebene abgeleitet, und anschließend mit ihren Vorversionen verglichen. Beide Vorgehensweisen führen zu denselben Ergebnissen, da die Transformation vollständig automatisiert durchgeführt wird.
Im weiteren Verlauf der Arbeit wird die Modelldifferenz nach dem Vorgehen des zweiten Falls ermittelt.

6.6.2 Erweitertes SOM-R-Vorgehensmodell

Das erweiterte Vorgehensmodell der SOM-R-Methodik definiert den Prozess zur Durchführung der Hauptphasen A, D und R als Lösungsverfahren der Systemweiterentwicklungsaufgabe. Abbildung 6.20 zeigt das detaillierte Vorgehensmodell und stellt neben den Entwicklungsschritten und -artefakten auch die funktionalen Beziehungen zwischen den eingesetzten

Hilfsmitteln, bzw. den Zugriff auf die bereitgestellten Informationen von Dokumentation und Modellvergleich sowie der Empfehlungsbasis dar.

Während die Aktivität Analyse (A1, A2) sowie die Ableitung von Handlungsempfehlungen (P1) in automatisierter Form beschreibbar sind, erfordert die Überarbeitung (Konsolidierung) des geänderten Schemas weiterhin die Einbeziehung des Systementwicklers, der bei der Bewältigung der Systemweiterentwicklungsaufgabe durch die Bereitstellung von *Assistenzinformationen* unterstützt werden. Die Bestandteile und der Aufbau des Vorgehensmodells sowie der Ablauf des Lösungsverfahrens werden im Folgenden erläutert.

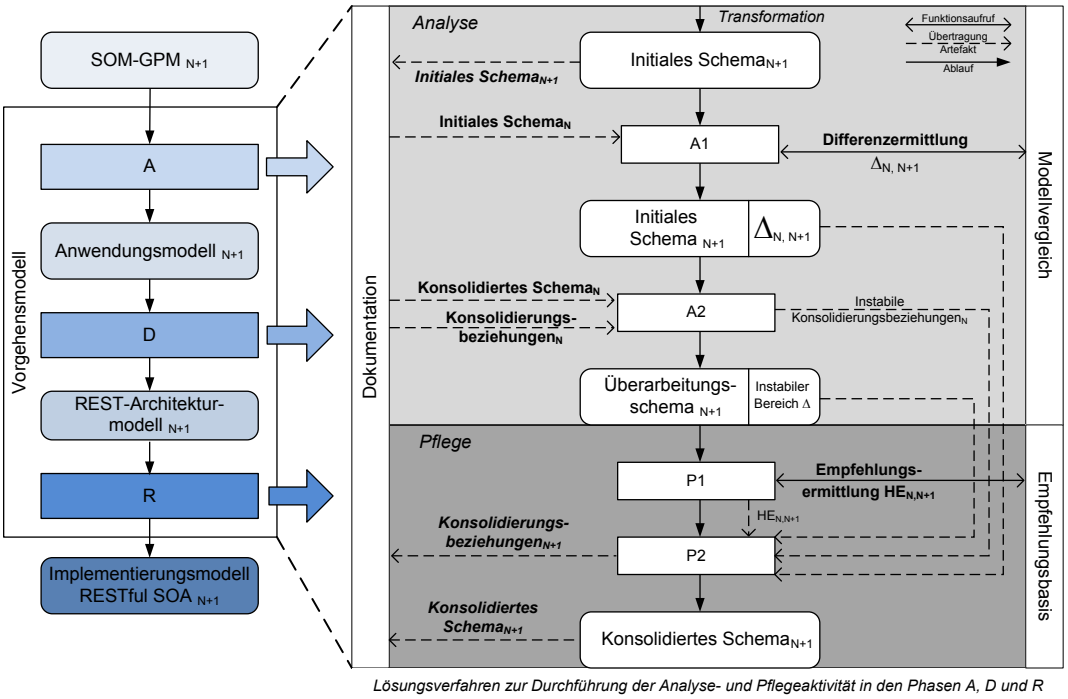


Abbildung 6.20: Erweitertes Vorgehensmodell der SOM-R-Methodik (detailliert)

LÖSUNGSVERFAHREN DER AKTIVITÄT ANALYSE

1. *Artefakt initiales Schema$_{N+1}$*: Die Analyseaktivität einer Modellebene wird durch das Vorliegen einer neuen Version ihrer initialen Schemata ausgelöst. Die neuen Schemaversionen werden gespeichert (abhängig von der gewählten Dokumentationsstrategie).

2. *Entwicklungsschritt A1 – Bestimmen der Modelldifferenz*: Ziel von A1 ist die Ermittlung des *Modell-Deltas* $\Delta_{N,\,N+1}$ zwischen der jeweils ursprünglichen und der angepassten Version der initialen Schemata einer Modellebene. Zu diesem Zweck werden die initialen Schemata$_N$ auf Basis der Dokumentation bereitgestellt und zusammen mit den initialen Schemata$_{N+1}$ die *Differenzermittlung* unter Einsatz des Hilfsmittels *Modellvergleich* durchgeführt.

3. Die *Artefakte initiales Schema$_{N+1}$* und *Modell-Delta $\Delta_{N,\,N+1}$* sind das Ergebnis von A1. Sie bilden den Input für Entwicklungsschritt A2.

4. *Entwicklungsschritt A2 – Ermitteln des Überarbeitungsschemas$_{N+1}$*: Der logische Ablauf (das Lösungsverfahren) zur Ermittlung des Überarbeitungsschemas besteht aus fünf Schritten:

   4.1 Bereitstellen des dokumentierten, konsolidierten Schemas$_N$.

   4.2 Bereitstellen der Konsolidierungsbeziehungen$_N$, die die Überführung der initialen Schemata$_N$ in die konsolidierten Schemata$_N$ einer Modellebene dokumentieren.

   4.3 Wiederhole für jeden Baustein des Modell-Deltas$_{N, N+1}$ (initiales Schema$_{N+1}$):

   - Ein Baustein des Modell-Deltas sowie seine Beziehungen zu (ebenfalls) geänderten Bausteinen werden dem instabilen Modellbereich des konsolidierten Schema$_N$ als neue Elemente hinzugefügt. Die Änderungsmarkierung wird übernommen.

   4.4 Wiederhole für jede Beziehung zwischen einem geänderten und einem unveränderten Baustein des Modell-Deltas$_{N, N+1}$:

   - Leite auf Basis der Konsolidierungsbeziehungen$_N$ für den unveränderten Baustein einer Beziehung den dazu korrespondierenden Baustein im konsolidierten Schema$_N$ ab. Aktualisiere die Beziehung zum abgeleiteten Baustein und füge sie dem instabilen Bereich des konsolidierten Schemas$_N$ hinzu.
   - Markiere die Konsolidierungsbeziehungen, die instabile Bausteine oder Bausteine, die auf Basis instabiler Bausteine abgeleitet wurden, enthalten.

   4.5 Markiere unveränderte Bausteine, die mit einer instabilen Beziehung verknüpft sind, als instabil (i).

   Das konsolidierte Schema$_N$ und der ermittelte instabile Modellbereich spezifizieren das *Überarbeitungsmodell$_{N+1}$* der Modellebene. Die Darstellung des instabilen Bereichs erfolgt in Form eines Modell-Deltas (Abschnitt 6.4.2.3). Das algorithmische Lösungsverfahren zur Bestimmung des Überarbeitungsschemas wird in Abschnitt 6.6.3 vorgestellt.

   Hinweis Implementierungsmodell: Die Schritte R.A1 und R.A2 werden auf der Modellebene der Implementierung zusammengefasst und gemeinsam durchlaufen (Abschnitt 6.6.1). Demnach stimmen auf dieser Ebene meist das initiale Schema$_{N+1}$ sowie das Modell-Delta$_{N,N+1}$ mit dem Überarbeitungsmodell$_{N+1}$ überein.

5. Das *Artefakt Überarbeitungsschema$_{N+1}$* ist der Output der Analyseaktivität und der zentrale Entwicklungsgegenstand auf dessen Grundlage die konsolidierten Schemata$_{N+1}$ der anzupassenden Modellebene durch den Systementwickler erstellt werden.

LÖSUNGSVERFAHREN DER AKTIVITÄT PFLEGE

6. *Entwicklungsschritt P1 – Ermitteln von Handlungsempfehlungen*: Die instabilen Bereiche des Überarbeitungsschemas$_{N+1}$ bilden den Input für die Ermittlung von Handlungsempfehlungen (HE). Für die Änderungen eines instabilen Schemaobjekts werden in P1 die Maßnahmen zu ihrer Behandlung im Rahmen der Konsolidierung empfohlen (Abschnitt 6.5).

7. *Entwicklungsschritt P2 – Konsolidieren des Überarbeitungsschemas*: Das erzeugte Überarbeitungsschema$_{N+1}$ wird angepasst und das dadurch konsolidierte Schema$_{N+1}$ der Modellebene gebildet:

   7.1 *Konsolidierung der instabilen Bereiche:* Die instabilen Bereiche des Überarbeitungsschemas$_{N+1}$ markieren dessen veränderte Elemente sowie die Beziehungen zwischen den unveränderten und veränderten Teilbereichen des Schemas. Diese Bereiche stehen im Fokus einer Durchführung der Weiterentwicklung und sind in das bestehende System zu integrieren.

   In der SOM-R-Methodik wird der Systementwickler bei der Durchführung des zweiten Pflegeschrittes (P2) durch die Bereitstellung von **Assistenzinformationen** unterstützt. Die Assistenzinformationen sind:

   - *Überarbeitungsschema$_{N+1}$* als Gegenstand der durchzuführenden Konsolidierung. Die instabilen Bereiche des Überarbeitungsschemas markieren die veränderten Bausteine, die im Zuge der Konsolidierung zu prüfen und ggf. zu überarbeiten sind.
   - *Initiales Schema$_{N+1}$ und ermitteltes Modell-Delta$_{N,N+1}$*.
     Zusammen mit den bereitgestellten Konsolidierungsbeziehungen$_N$ können die getroffenen Entwurfsentscheidungen zur Bildung des konsolidierten Schemas$_N$ nachvollzogen werden.
   - *Markierte Konsolidierungsbeziehungen$_N$* von instabilen Bausteinen kennzeichnen potenziell ungültige Entwurfsentscheidungen (siehe A2, Schritt 4). Sie sind deshalb in P2 bezüglich einer Konsolidierung zu prüfen.
   - Menge von *Handlungsempfehlungen.*

   Die Durchführung der pflegenden Schema-Konsolidierung wird in Abschnitt 6.6.4 genauer vorgestellt.

   7.2 Ergebnis der Weiterentwicklung (P2) sind die konsolidierten Schemata$_{N+1}$ und die aktualisierten Konsolidierungsbeziehungen$_{N+1}$ der Modellebene. Die aktuellen Versionen der Entwicklungsartefakte und -entscheidungen werden dokumentiert.

8. *Artefakt konsolidiertes Schema$_{N+1}$*: Ergebnis der modellbasierten Pflege des Modellsystems sind die konsolidierten Schemata$_{N+1}$ der jeweils angepassten Modellebene. Sie werden archiviert. Die konsolidierten Schemata$_{N+1}$ bilden den Ausgangspunkt für die Ableitung der initialen Schemata$_{N+1}$ auf der nachfolgenden Modellebene im Entwicklungsprozess.

Das konzipierte Lösungsverfahren der Systemweiterentwicklungsaufgabe besitzt den Automatisierungsgrad *teilautomatisiert*. Die Aufgaben zur Bereitstellung von Assistenzinformationen (A1, A2, P1), welche die Ermittlung von Modelldifferenzen, Ableitung von Überarbeitungsschemata sowie Bestimmung von Handlungsempfehlungen umfassen, sind *automatisierbar*. Diese Schritte sind formal beschreibbar und somit durch den Einsatz geeigneter Werkzeuge, welche die Funktionalität von Aktivitäten und Hilfsmitteln realisieren, maschinell durchführbar. Die Aufgabe zur Spezifikation des Modellsystems selbst ist *nicht automatisierbar,* da im

Rahmen der pflegenden Schema-Konsolidierung vom Systementwickler Entwurfsentscheidungen zu treffen sind. P2 entspricht somit einer nicht-automatisierbaren Entscheidungsaufgabe [FeSi13, S. 37 ff.].

6.6.3 Erzeugung des Überarbeitungsschemas

Das Erzeugen des Überarbeitungsschemas spezifiziert das Lösungsverfahren von A2 in algorithmischer Form. Es dient damit als Basis für eine werkzeuggestützte Realisierung der Analyseaktivität. Korrespondierend zur Konzeption der methodischen Hilfsmittel der SOM-R-Methodik erfolgt auch die Formulierung des Algorithmus unabhängig vom Einsatz einer konkreten Technologie oder Programmierplattform. Eine natürlichsprachliche Beschreibung des Lösungsverfahrens wurde bereits im vorausgegangenen Abschnitt 6.6.2 eingeführt.

Algorithmus – Ableitung des Überarbeitungsschemas

Input:

- Menge $K = O_K \cup B_K$, als Vereinigung der Mengen der Objekte O und Beziehungen B des konsolidierten $Schemas_N$.
- Menge $I = O_I \cup B_I$, als Vereinigung der Objekte und Beziehungen des initialen $Schemas_{N+1}$.
- Menge $D = +/- O_D \cup +/- B_D$, als Vereinigung der veränderten (hinzugefügte/entfernte) Objekte und Beziehungen des Modell-$Deltas_{N,N+1}$ (Differenz von initialem $Schema_N$ und initialem $Schema_{N+1}$).
- Menge E = T , als Traces der $Konsolidierungsentscheidungen_N$. Jeder Trace besteht aus einer Menge von Vorgängerobjekten $O_V \times$ Abhängigkeitsbeziehung $B_A \times$ Menge von Nachfolgerobjekten O_N.

Output:

- Menge $Ü = O_Ü \cup B_Ü$ als vereinigte Objekte und Bez. des $Überarbeitungsschemas_{N+1}$.
- Menge E = T als Traces der $Konsolidierungsentscheidungen_{N+1}$ mit Instabilitätsmarkierung.

	Lösungsverfahren:	*Anmerkungen*
	// neu hinzugefügte Objekte	
1	**For Each** X = $+O_D$ *in* D **Do**	Wdhl. für alle X=+Obj. des Modell-Deltas:
2	**For Each** $+B_D(X, O) \vee +B_D(O, X)$ *in* D **Do**	Für jede +B, die X mit einem O verknüpft:
3	**If** O ***Not*** (Element of $+O_D$) *in* D **Then**	*Wenn* O nicht Element von D, *dann* …
4	Y = O	Y = O (ist es ein unverändertes Objekt O)
5	**For Each** (Element of Y) = O_V *of* T *in* E	Wiederhole für jeden Trace bei dem das
6	**Do**	Element von Y ein Vorgängerobjekt O_V ist:
7	*Set* Elem(Y) = O_N *of* T (O_V)	Bestimme die Menge der abhängigen
8	*Mark as* (i) T *in* E	Objekte O_N des konsol. Schemas anhand
9	**Loop**	der Traces in E und füge sie Y hinzu.
		Markiere die Traces als instabil (i).
10	**For Each** (Element of Y) **Do**	Wiederhole für jedes Element von Y:
11	*Mark as* (i) *every* O = Elem(Y) *in* K	Markiere die Objekte von Y als instabil (i)
12	$K \cup X$	in K. Füge X = +Objekt in die Menge K ein.
13	$K \cup +B_D(X, Elem(Y)) \vee +B_D(Elem(Y), X)$	Füge +B(X, Elem(Y)) in K ein.
14	**Loop**	
15	**Else**	
16	$K \cup (+B_D(X, O) \vee +B_D(O, X))$	… *Sonst* füge +B in K ein.
17	**End If**	
18	**Loop**	
19	**Loop**	

	// entfernte Objekte	
20	**For Each** X = $-O_D$ in D **Do**	Wiederhole für alle X=(–)Objekte des Modell-Deltas:
21	Y = X	
22	**For Each** (Element of Y) = O_V *of* T *in* E **Do**	Bestimme für jeden Trace mit einem Elem. von Y als O_V die Menge abhängiger Objekte O_N anhand der Traces. Markiere die Traces als instabil (i).
23	*Set* Elem(Y) = O_N *of* T (O_V)	
24	*Mark as* (i) T *in* E	
25	**Loop**	
26	*Mark as* (–) every O = Elem(Y) *in* K	Markiere jedes Objekt O von Y als (–) in K.
27	**For Each** $-B_D$ (X, O) $\vee$ $-B_D$ (O, X) *in* D **Do**	Für jede (–)B, die X mit O verknüpft:
28	**If** O ***Not*** (Element of $-O_D$) *in* D **Then**	*Wenn* O nicht entfernt wurde, *dann …*
29	Y = O	(ist O ein unverändertes Objekt)
30	**For Each** (Element of Y) = O_V *of* T *in* E	Bestimme für jedes Elem. von Y die
31	**Do**	Menge O_N der von O_V abhängigen Objekte
32	*Set* Elem(Y) = O_N	des konsol. Schemas anhand der Traces in
33	**Loop**	E. Füge O_N in Y ein.
34	**For Each** (Element of Y) **Do**	
35	*Mark as* (i) O = (Element of Y) *in* K	Markiere die Objekte von Y als instabil (i)
36	*Mark as* (–) B(X, Y) $\vee$ B (Y, X) *in* K	in K. Markiere die Beziehung B in K als (–).
37	**Loop**	
38	**Else**	… *Sonst* (ist O als „verändert“ markiert)
39	*Mark as* (–) B(X, Y) $\vee$ B (Y, X) *in* K	Markiere die Beziehung B in K als (–)
40	**End If**	
41	**Loop**	
42	**Loop**	
43	Ü = K	

Legende:

Markierung M := {keine, (i) Instabilität, (+) hinzugefügt, (–) entfernt}

X, Y := Variablen — *Set* (X = Y) := Setze Menge X = Y

(Elem(X)) := ein Element der Menge X — *Mark as* (M):= Markiere das Objekt mit (M)

Der erste Teil des Algorithmus (Zeilen 1-19) spezifiziert das Lösungsverfahren, um die neu hinzugefügten Objekte und Beziehungen des initialen $Schema_{N+1}$ in das konsolidierte $Schema_N$ zu überführen (**For Each** X = $+O_D$ *in* D **Do**). Hierzu werden auf Basis der dokumentierten Traces T für jede Beziehung, die ein hinzugefügtes mit einem unveränderten Objekt des initialen $Schemas_{N+1}$ verknüpft, die aus dem unveränderten Objekt gebildeten Objekte des konsolidierten $Schemas_N$ bestimmt. Dies erfolgt durch Ermittlung der Menge von O_N auf Basis derjenigen Traces, in denen das unveränderte Objekt ein Vorgängerobjekt O_V ist (5-9). Die identifizierten (unveränderten) Objekte des konsolidierten $Schemas_N$ werden zudem als instabil markiert (*Mark as* (i) *every* O = Elem(Y) *in* K). Auch die korrespondierenden Traces werden als instabil markiert (*Mark as* (i) T *in* E). Abschließend werden alle hinzugefügten Objekte, sowie die hinzugefügten Beziehungen (mit den aktualisierten Objektreferenzen) in das konsolidierte $Schema_N$ eingefügt (K $\cup$ X und K $\cup$ $+B_D$ (X, Elem(Y)) $\vee$ $+B_D$ (Elem(Y), X), Zeilen 12-13). Entsprechend diesem Vorgehen werden im zweiten Teil des Algorithmus die Objekte und Beziehungen, die im initialen $Schema_{N+1}$, als „entfernt“ markiert wurden, in das konsolidierte $Schema_N$ überführt und eingefügt (Zeilen 20-42). Das Ergebnis des Lösungsverfahrens ist das $Überarbeitungsschema_{N+1}$ (Ü=K, 43).

Die Anwendung des Algorithmus wird am Beispiel eines KOS gezeigt. Abbildung 6.21 fasst die Ergebnisse zusammen. Jedes geänderte Schemaobjekt im initialen KOS_{N+1} wird auf Grundlage des ermittelten Modell-Deltas (grafisch dargestellt), des konsolidierten KOS_N und der Konsolidierungsbeziehungen$_N$ in das KOS-Überarbeitungsschema$_{N+1}$ überführt. Alle neu hinzugefügten Objekte (+*Bestätigung*) werden übernommen und mit den unveränderten Objekten des konsolidierten KOS_N in Beziehung gesetzt. Die hinzugefügte interacts_with-Beziehung zwischen *Auftrag* und *Bestätigung* wird korrespondierend zur Zerlegung von *Auftrag* in *Auftragskopf* und *Auftragsposition* in Form zweier interacts_with-Beziehungen in das Überarbeitungsschema$_{N+1}$ eingefügt. Die Traces (Konsolidierungsbeziehungen) von *Auftrag* sind instabil. Die unveränderten Objekttypen *Auftragskopf*, *Auftragsposition* und *Lieferung* stehen mit geänderten Bausteinen in Beziehung und werden mit einer Instabilitätsmarkierung versehen.

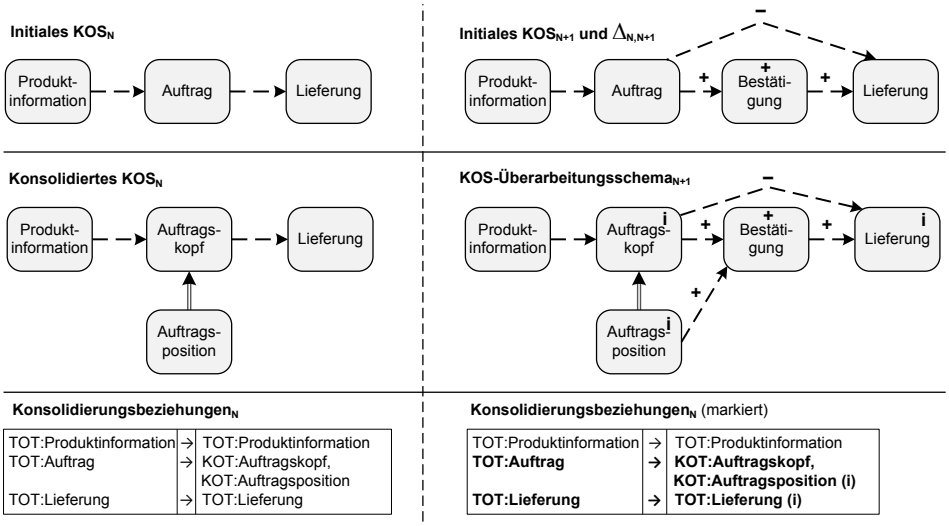


Abbildung 6.21: Bestimmung des Überarbeitungsschemas (Beispiel KOS)

6.6.4 Durchführung der pflegenden Schema-Konsolidierung

Im letzten Schritt P2 der Systempflege werden die erzeugten Überarbeitungsschemata$_{N+1}$ vom personellen Aufgabenträger *Systementwickler* überarbeitet und so die konsolidierten Schemata$_{N+1}$ der Modellebene erstellt. Der *instabile Bereich* des Überarbeitungsschemas steht hier im Fokus der Entwicklung. Die bereitgestellten *Assistenzinformationen* unterstützen die personelle Durchführung der Entwicklungstätigkeit (siehe Abschnitt 6.6.2). Das Zusammenspiel zwischen der zu Schritt P2 korrespondierenden Aufgabe, dem Aufgabenträger und den unterstützenden Artefakten veranschaulicht Abbildung 6.22.

Während die Assistenzinformationen in automatisierter Form erzeugbar sind, erfolgt die Konsolidierung des Überarbeitungsschemas durch den Systementwickler (personelle Aufgabe). Dieser wiederum kann bei der Durchführung der Entscheidungsaufgabe (P2) ein Entwicklungswerkzeug nutzen. Neben dem konsolidierten Schema$_{N+1}$ werden auch die zugehörigen Konsolidierungsbeziehungen in ihrem aktualisierten Stand als Output der Aufgabe dokumentiert (Version N+1).

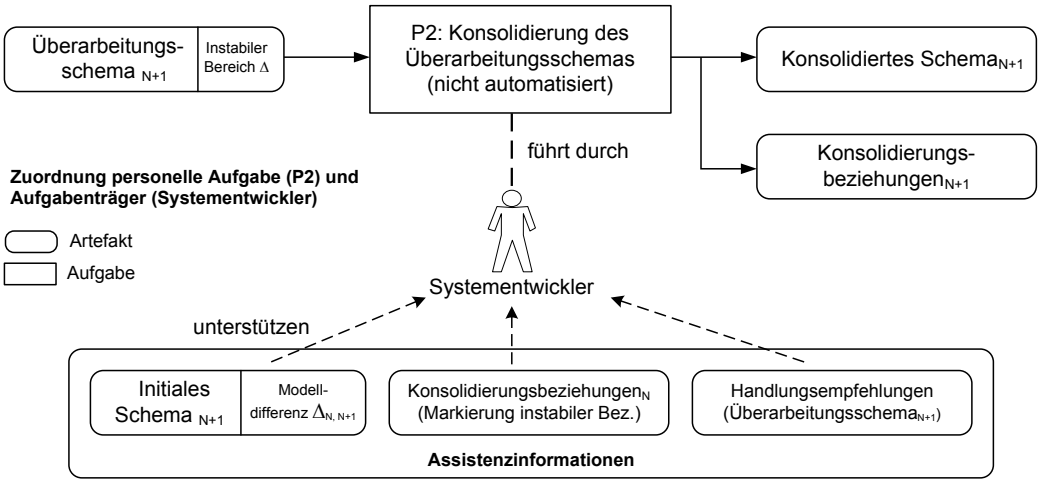


Abbildung 6.22: Durchführung der Aufgabe *pflegende Schema-Konsolidierung* (P2)

Die pflegende Konsolidierung der Ebenen Anwendungsmodell, REST-Architekturmodell und Implementierungsmodell des SOM-R-Modellsystems unterscheidet sich primär hinsichtlich der definierten Schemata und ihrer Metaobjekte (Sichten des integrierten Metamodells). Ihre Durchführung wird nachfolgend genauer betrachtet.

Bestimmung von Operationen zur Konsolidierung der Überarbeitungsschemata

Bei der Weiterentwicklung eines Überarbeitungsschemas werden dessen Objekte bzw. Bausteine durch den Systementwickler entfernt und/oder neu hinzugefügt. Zudem werden die Eigenschaften bereits existierender Schemaobjekte aktualisiert, um z. B. die Kommunikation mit neu hinzugefügten Objekten realisieren zu können. Das Anpassen eines existierenden Bausteins entspricht dabei dem Entfernen der ursprünglichen und dem Hinzufügen der neuen Version eines Bausteins mit aktualisierten Eigenschaften. *Anpassen/Aktualisieren* ist somit als Abfolge der beiden elementaren Änderungsoperationen definiert und wird als zusammengesetzte Operation verstanden (Abschnitt 6.4.1).

Das Hinzufügen eines angepassten, neuen Bausteins zieht jedoch ein Entfernen und Hinzufügen neuer Beziehungen zu seinen benachbarten Bausteinen nach sich. Die Eigenschaften dieser Bausteine wären dann ebenfalls an die veränderte Beziehung anzupassen. Es besteht somit die Gefahr einer *Fortpflanzung* der durchzuführenden Schemaänderungen und damit der dynamischen Ausbreitung des instabilen Bereichs auf das gesamte Überarbeitungsschema.

Zur Lösung des dargestellten Problems der Änderungsfortpflanzung wird nachfolgend die Operation *Anpassen*, neben Hinzufügen und Entfernen, als dritte (zusammengesetzte) Operation definiert, um die Ausbreitung des instabilen Bereichs bei der Konsolidierung des Überarbeitungsschemas zu begrenzen. Die veränderten Bausteine des Überarbeitungsschemas sind demnach grundsätzlich drei Mengen zuordenbar:

- *Menge der neu hinzugefügten Bausteine.*
- *Menge der entfernten Bausteine.*
- *Menge angepasster bestehender Bausteine* (neue Version N+1).

Die Bildung der Menge von angepassten Bausteinen ermöglicht die Abstimmung der instabilen Objekte im unveränderten, mit dem geänderten Bereich des Überarbeitungsschemas. Die bisher unveränderten Schemaobjekte, deren Eigenschaften im Zuge der Anpassung aktualisiert wurden, erhalten eine neue Objektversion (Abschnitte 6.5.2.2.3 und 6.5.2.3.3). Durch die Aktualisierung der Version eines angepassten Objekts bleibt dessen Objektname stabil. Folglich entsteht bezüglich der Verknüpfung zu weiteren unveränderten Objekten kein *direkter* Bedarf zur Änderung des Lösungsverfahrens in diesen Objekten. Für die Nachrichten und Methoden der mit der neuen Version eines aktualisierten Objekts verknüpften Schemaobjekte des stabilen Schemabereichs ist die Notwendigkeit einer Aktualisierung somit zunächst zu prüfen und nur bei Bedarf vorzunehmen. Durch die gesonderte Behandlung der „Operation Anpassen“ wird die Gefahr einer Fortpflanzung von Änderungsmarkierungen und damit die Ausweitung des instabilen Schemabereichs reduziert.

Abbildung 6.23 veranschaulicht den vorgestellten Lösungsansatz am Beispiel der Weiterentwicklung eines KOS. Der betrachtete Ausschnitt des KOS-Überarbeitungsschemas$_{N+1}$ zeigt die zu behandelnden Änderungen sowohl in grafischer als auch tabellarischer Form. Weitere Assistenzinformationen sowie die überarbeiteten Konsolidierungsbeziehungen$_{N+1}$ werden nicht dargestellt.

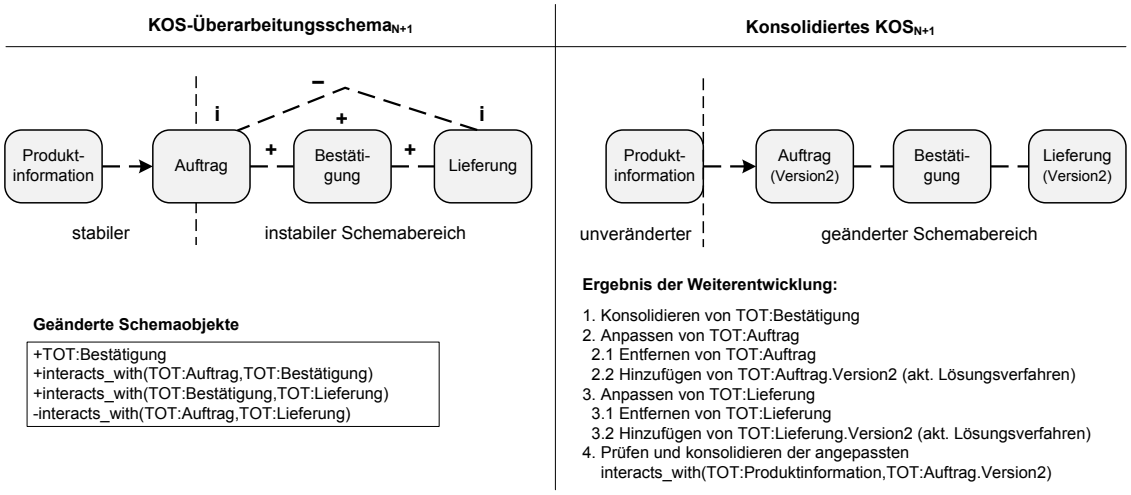


Abbildung 6.23: Änderungsarten der pflegenden Schema-Konsolidierung (Beispiel KOS)

Die geänderten Objekte des KOS-Überarbeitungsschemas$_{N+1}$ sind der hinzugefügte Objekttyp *TOT:Bestätigung* und dessen Interaktionsbeziehungen zu benachbarten Objekttypen. Weiter wurde die Interaktionsbeziehung zwischen *TOT:Auftrag* und *TOT:Lieferung* entfernt. Die entfernte Beziehung wird zur Verdeutlichung der Änderungen weiterhin visualisiert. Die unveränderten Objekttypen *TOT:Auftrag* und *TOT:Lieferung* werden als instabil markiert. Die Bausteine des stabilen Schemabereichs sind somit *TOT:Produktinformation* und die Interaktionsbeziehung zu *TOT:Auftrag*. Das konsolidierte KOS$_{N+1}$ ist das Ergebnis der Durchführung von Aufgabe P2. Der neu hinzugefügte *TOT:Bestätigung* wird konsolidiert. Ein Zusammenfassen mit weiteren konzeptuellen Objekttypen oder ein Zerlegen erfolgt nicht.

Die Behandlung der beiden instabil markierten Bausteine (*TOT:Auftrag*, *TOT:Lieferung*) beschränkt sich auf das Anpassen ihrer Eigenschaften zur Realisierung der geänderten Beziehungen zu dem hinzugefügten Baustein. Entsprechend dem eingangs dargelegten

Vorgehen wird die Aktualisierung mittels Einfügen neuer Baustein-Versionen (*TOT:Auftrag.Version2* und *TOT:Lieferung.Version2*) realisiert. Das Anpassen von *TOT:Auftrag* zieht das Entfernen der ursprünglichen interacts_with-Beziehung zu *TOT:Produktinformation* und das Hinzufügen einer neuen Beziehung nach sich. Die Beziehung ist entsprechend den Regeln der Konsolidierung zu überarbeiten. Für *TOT:Produktinformation* ist zu prüfen, ob die aktualisierten Eigenschaften des *Auftrags* einen weiteren Überarbeitungsbedarf ergeben. Da der Objektname von *Auftrag* nicht geändert wurde, bestehen keine direkten Auswirkungen auf die Kommunikation mit dem Objekttyp der *Produktinformation*.

6.6.5 Beispielhafte Anwendung des Lösungsansatzes der Systempflege

Die Anwendung des konzipierten Lösungsansatzes der SOM-R-Methodik zur Systempflege wird am Beispiel der Fallstudie *Abwicklung von Flugbuchungen* demonstriert. Auslöser für die Durchführung der Systemweiterentwicklung sind die Strukturänderungen am flexiblen SOM-Geschäftsprozessmodell$_1$ (Version N=1). Anhand zweier Szenarien wird die Anwendung der SOM-R-Methodik zur Pflege des spezifizierten Informationssystems erläutert.

6.6.5.1 Erstes Szenario – Einbindung eines externen Dienstleisters

Die erste Änderung erweitert den vorliegenden Geschäftsprozess im Rahmen der Integration eines externen Dienstleisters und dem Einfügen von Beziehungen zu seiner Koordination. Konkret sollen dem Kunden zukünftig nach einer Flugbuchung aktuelle Informationen zu dessen Flugbuchung bereitgestellt werden. Diese Aufgabe führt die Fluglinie nicht selbst durch, sondern erteilt einen Serviceauftrag an einen *Fluginformationsservice* (FIS) als externen Dienstleister. Nach Abschluss der Informationsbereitstellung erhält die Fluglinie eine Zusammenfassung der erbrachten Serviceleistung. Den relevanten Ausschnitt der neuen Version (N+1 = 2) des SOM-Geschäftsprozessmodells$_2$ zeigt Abbildung 6.24.

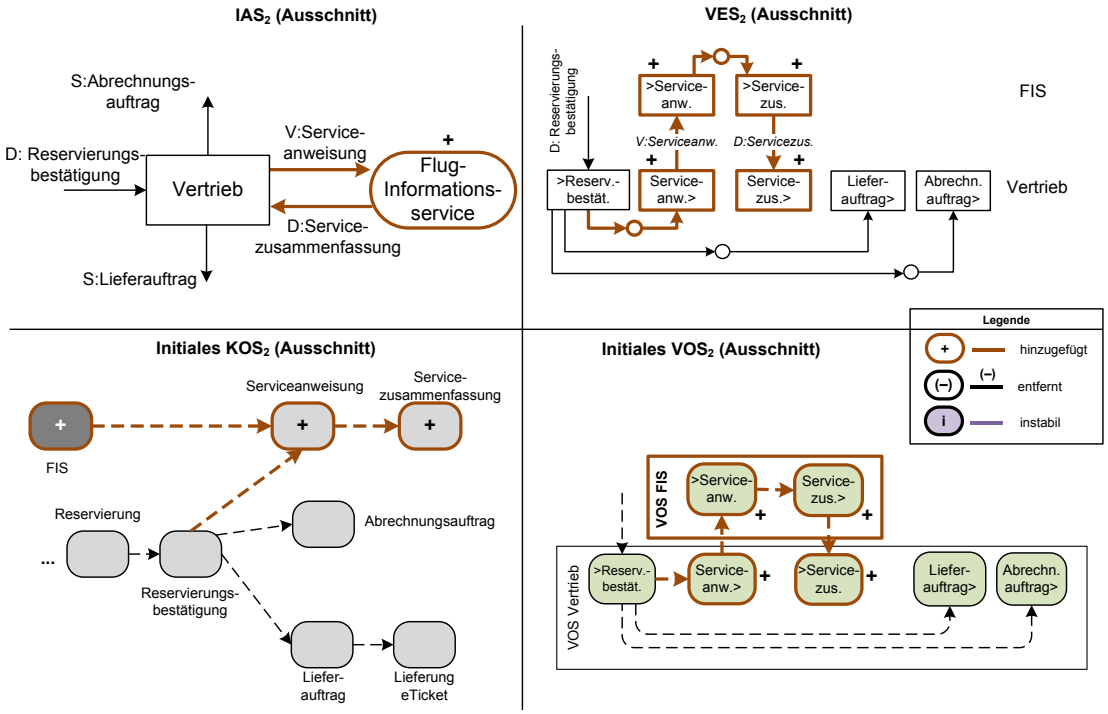


Abbildung 6.24: Neue Version (2) von SOM-GPM und initialer AwS-Spezifikation (Szenario FIS)

Das SOM-Geschäftsprozessmodell$_2$ wird im strukturorientierten IAS um ein betriebliches Objekt *Fluginformationsservice* sowie die Transaktionen *V:Serviceanweisung* und *D:Servicezusammenfassung* erweitert. Aus verhaltensorientierter Sicht (VES) führen zwei neue Aufgaben des Vertriebes die Koordination des FIS durch. Ein objektinternes Ereignis verknüpft die Aufgabe *>Reservierungsbestätigung* mit der Aufgabe *Serviceanweisung>* und legt damit die Ausführungsreihenfolge zwischen diesen fest. Die angepassten Schemata IAS$_2$ und VES$_2$ werden im Rahmen des Transformationsschrittes T1 in die neuen Versionen des initialen KOS$_2$ und initialen VOS$_2$ überführt, die der Input der ersten Analyseaktivität (A.A1) sind.

Das Ergebnis des Modellvergleichs ist das ermittelte Modell-Delta$_{1,2}$ der initialen AwS-Schemata KOS und VOS, das Tabelle 6.18 in der vereinfachten Form darstellt. Um die Auswirkungen der einzelnen Änderungen der vorliegenden Strukturflexibilität zu verdeutlichen, wird auch die Modelldifferenz auf GP-Ebene bestimmt (G.A1). Die gewählte tabellarische Darstellung dient der übersichtlichen Repräsentation der Ergebnisse. Die Beziehungen zwischen den Modellbausteinen sind den verknüpfenden Bausteinen in Klammern „()“ zugeordnet (vgl. Tabelle 6.1 und Tabelle 6.2).

Schema	Geändertes Modellobjekt
IAS	+ bO:Fluginformationsservice
IAS (VES)	+ bT:Serviceanweisung(bO:Vertrieb, bO:FIS)
IAS (VES)	+ bT:Servicezusammenfassung(bO:FIS,bO:Vertrieb)
VES	+ A:Serviceanweisung>
VES	+ A:>Serviceanweisung
VES	+ A:Servicezusammenfassung>
VES	+ A:>Servicezusammenfassung
VES	+ iE(A:>Reservierungsbestätigung, A:Serviceanweisung>)
VES	+ iE(A:>Serviceanweisung, A:Servicezusammenfassung>)
iKOS	+ OOT:Fluginformationsservice
iKOS	+ TOT:Serviceanweisung
iKOS	+ TOT:Servicezusammenfassung
iKOS	+ interacts_with(TOT:Reservierungsbest.,TOT:Serviceanweisung)
iKOS	+ interacts_with(TOT:Serviceanweisung,TOT:Servicezusammenf.)
iKOS	+ interacts_with(TOT:Fluginformationsservice,TOT:Serviceanweisung)
iVOS	+ VOT:Serviceanweisung>(VOS:Vertrieb)
iVOS	+ VOT:>Serviceanweisung(VOS:Fluginformationsservice)
iVOS	+ VOT:Servicezusammenfassung>(VOS:Fluginformationsservice)
iVOS	+ VOT:>Servicezusammenfassung(VOS:Vertrieb)
iVOS	+ interacts_with(VOT:>Reservierungsbestätigung,VOT:Serviceanweisung>)
iVOS	+ interacts_with(VOT:Serviceanweisung>,VOT:>Serviceanweisung)
iVOS	+ interacts_with(VOT:>Serviceanweisung,VOT:Servicezusammenf.>)
iVOS	+ interacts_with(VOT:Servicezusammenf.>,VOT:>Servicezusammenf.)

Tabelle 6.18: Modell-Delta$_{1,2}$ von SOM-GPM und AwS-Spezifikation (Szenario FIS)

Die initialen Schemata$_2$ der fachlichen AwS-Spezifikation und das ermittelte Modell-Delta$_{1,2}$ bilden den Input für die Durchführung der zweiten Analyseaktivität (A.A2), welche die Ermittlung der Überarbeitungsschemata der Modellebene zum Ziel hat. Als weitere Informationen fließen die entwickelten Vorversionen der konsolidierten Schemata$_1$ und Konsolidierungsbeziehungen$_1$ der fachlichen AwS-Spezifikation in die Analyse ein (vgl. Abbildung 5.13 und Tabelle 6.6). Die instabilen Konsolidierungsbeziehungen werden auf Basis der verknüpften unveränderten Bausteine im Modell-Delta bestimmt, die auch eine Instabilitätsmarkierung

erhalten. Dies sind im vorliegenden Fall die Bausteine *TOT:Reservierungsbestätigung* und *VOT:Reservierungsbestätigung>.* Die betroffenen Konsolidierungsbeziehungen zeigt Tabelle 6.19.

	V	B	N
	...	...	...
(i)	TOT:Reservierungsbestätigung	→	TOT:Reservierungsbestätigungskopf, TOT:Reservierungsbestätigungsposition
(i)	TOT:Reservierungsbestätigungskopf, TOT:X_kopf	→	TOT:Auftragskopf
(i)	TOT:Reservierungsbestätigungsposition, TOT:X_position	→	TOT:Auftragsposition
(i)	VOT:>Reservierungsbestätigung	→	VOT:>Reservierungsbestätigung *(keine Änderung)*
	...	...	...

X = {Wahl, Angebot, Angebotsbestätigung, Auftragszusammenfassung, Buchung, Reservierung, [Reservierungsbestätigung], Buchungsbestätigung}

Tabelle 6.19: Instabile Konsolidierungsbeziehungen$_1$ der fachlichen AwS-Spezifikation (Szenario FIS)

In der vorangegangenen Entwicklung (Version 1) wurde bspw. der *TOT:Reservierungsbestätigung* zunächst in einen Kopf und seine Positionen zerlegt, und anschließend mit den weiteren transaktionsspezifischen Objekttypen (X) zu einem *Auftragskopf* bzw. den *Auftragspositionen* zusammengefasst. Die von dem instabilen TOT abhängigen Konsolidierungsbeziehungen$_1$ sind deshalb als instabil zu markieren. *VOT:>Reservierungsbestätigung* wurde hingegen nicht überarbeitet, wird jedoch zur vollständigen Darstellung der instabilen Beziehungen aufgeführt.

Auf Grundlage des beschriebenen Inputs erfolgt die Bildung der Überarbeitungsschemata$_2$ von KOS und VOS nach dem Lösungsansatz von Schritt A.A2 in der SOM-R-Methodik (vgl. Erläuterung zu A.A2 in den Abschnitten 6.6.2 und 6.6.3). Die hinzugefügte interacts_with-Beziehung zwischen *TOT:Serviceanweisung* und *TOT:Reservierungsbestätigung* (initiales KOS$_2$) wird in das konsolidierte KOS$_1$ überführt und verknüpft dort *KOT:Auftragskopf* und *KOT:Auftragspositionen* mit *TOT:Serviceanweisung* (vgl. Konsolidierungsbeziehungen$_1$). Auftragskopf und Auftragsposition sind somit ebenfalls als instabil zu markieren. Das VOS-Überarbeitungsmodell$_2$ umfasst das konsolidierte VOS$_1$ sowie die vier hinzugefügten VOTs zur Koordination der Servicedienstleistung. Die interacts_with-Beziehung zwischen *VOT:Serviceanweisung>* und dem instabilen *VOT:>Reservierungsbestätigung* bleibt unverändert. Abbildung 6.25 zeigt die Überarbeitungsschemata$_2$ von KOS bzw. VOS als Output der zweiten Analyseaktivität (A.A2).

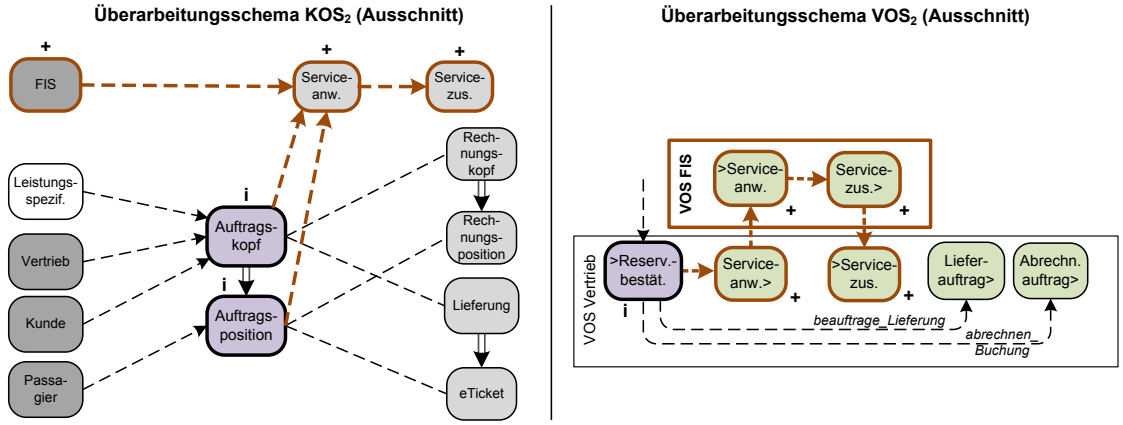


Abbildung 6.25: Überarbeitungsschemata$_2$ von KOS und VOS (Szenario FIS)

Die beiden Überarbeitungsschemata sind der Input der ersten Pflegeaktivität (A.P1) zur Ermittlung von Handlungsempfehlungen. Ausgehend von den geändert und instabil markierten Bausteinen der Überarbeitungsschemata$_2$ werden die Änderungstypen des Anwendungsmodells bestimmt und konkrete Empfehlungen abgeleitet (Abschnitt 6.5.1). Tabelle 6.20 zeigt eine Teilmenge der Handlungsempfehlungen.

Beispielsweise gelten für die Überarbeitung des *+TOT:Serviceanweisung* die Schritte der Konsolidierung K1.1. Die „Platzhalter" in den Auswirkungen von Änderungstypen werden mit den geänderten Objekten des KOS-Überarbeitungsschema$_2$ konkretisiert und bilden so die zu bearbeitenden Änderungsinstanzen. Im KOS-Schema sind z. B. *KOT:Auftragskopf* und *KOT:Auftragsposition* die Objekttypen mit Instabilitätsmarkierung. Die VOTs von *VOS Vertrieb* (und dem externem *VOS:FIS*) können auf die hinzugefügten KOTs zugreifen. Entsprechend sind ihre Lösungsverfahren hinsichtlich einer Berücksichtigung dieser Änderung (Vorliegen neuer KOTs mit ihren Eigenschaften) zu prüfen und ggf. anzupassen.

Änderungsinstanz	Handlungsempfehlung
+ TOT:Service-anweisung	Überarbeite +TOT:Serviceanweisung nach den Schritten der Konsolidierung K1.1.
	• Prüfe die Eigenschaften verknüpfter instabiler Objekttypen (KOT:Auftragskopf, KOT:Auftragsposition) bzgl. des neu verknüpften TOT:Serviceanweisung. • Prüfe ein Konsolidieren der instabilen Objekttypen (KOT:Auftragskopf, KOT:Auftragsposition) und der unveränderten Objekttypen (LOT:Leistungsspezifikation, OOT:Vertrieb, […], TOT:eTicket) • Prüfe das Anpassen von VOT-Lösungsverfahren in (VOS Vertrieb) im Hinblick auf +TOT:Serviceanweisung
(i) KOT:Auftragskopf	*Anpassen der Eigenschaften* von KOT:Auftragskopf zur Realisierung der Interaktion (Empfang/Senden von +/-Nachrichten) mit +TOT:Serviceanweisung
	⇒ Prüfe die verknüpften Objekttypen (Leistungsspezifikation, Vetrieb, Kunde, …)
+ VOT:Service-anweisung>	Überarbeite +VOT:Serviceanweisung> nach den Schritten der Konsolidierung K1.2.
	• Prüfe die Zuordnung und Konsolidierung von IOT • Prüfe die VOT-Lösungsverfahren bezüglich +VOT:Serviceanweisung> • Prüfe ein Konsolidieren von instabilen Objekttypen (VOT:>Reservierungsbest.)
…	…

Tabelle 6.20: Ausschnitt aus den ermittelten Handlungsempfehlungen$_2$ (Szenario FIS)

In der zweiten Pflegeaktivität (A.P2) führt der Systementwickler die Anpassung des Anwendungsmodells durch (Abschnitt 6.6.4). Zur Unterstützung der Weiterentwicklungstätigkeit werden ihm die erzeugten Assistenzinformationen an die Hand geben. Einen Ausschnitt des konsolidierten KOS$_2$ und VOS$_2$ als Ergebnis der pflegenden Schema-Konsolidierung des Anwendungsmodells zeigt Abbildung 6.25.

Im KOS-Überarbeitungsschema werden *TOT:Serviceanweisung* und *TOT:Sevicezusammenfassung* im Zuge der Konsolidierung K1.1 zu einem *KOT:Serviceauftrag* zusammengefasst. Der KOT weist somit zwei Ausführungszustände ('*Service_angewiesen*', '*Serviceleistung_zusammengefasst*') auf. Der *OOT:FIS* wird konsolidiert und in *KOT:Servicespezifikation* umbenannt. Die Eigenschaften des *KOT:Auftragskopf* werden zur Realisierung der neuen Beziehung zu *KOT:Serviceauftrag* überarbeitet und *KOT:Auftragskopf* mit der neuen Version (2) hinzugefügt.

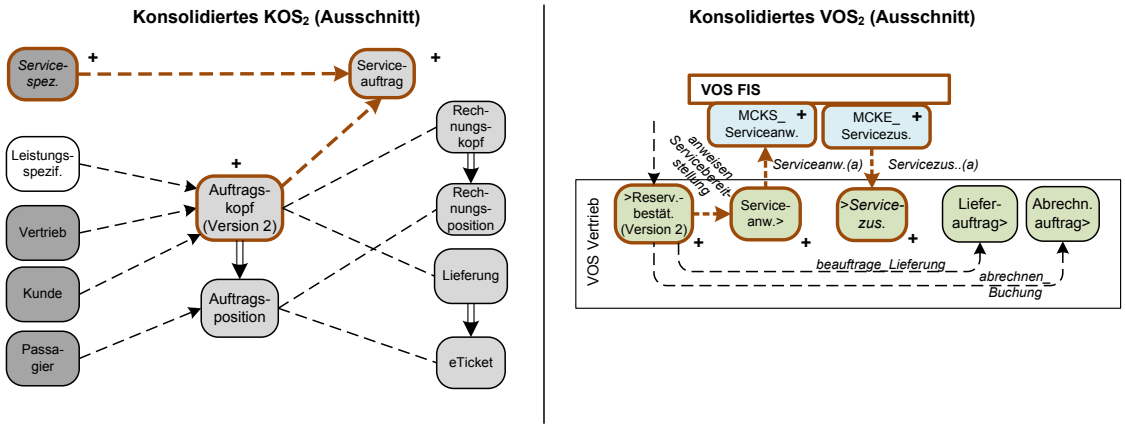


Abbildung 6.26: Ausschnitt des konsolidierten KOS_2 und des konsolidierten VOS_2 (Szenario FIS)

Die Konsolidierung des VOS sieht das Entfernen der nicht-automatisierten VOTs des externen *VOS Fluginformationsservice* vor. Zur Realisierung der Kommunikationsbeziehung mit dem externen Dienstleister werden *IOT:MCKS_Serviceanweisung* und *IOT:MCKE_Servicezusammenfassung* hinzugefügt. Der aktualisierte Stand der Konsolidierungsbeziehungen$_2$ wird parallel zur Überarbeitung dokumentiert (Tabelle 6.21).

V	B	N	A	I
…	…	…		
TOT:Serviceanweisung, TOT:Servicezusammenfassung	→	KOT:Serviceauftrag	2	
OOT:Fluginformationsservice	→	KOT:Servicespezifikation	2	
TOT:Reservierungsbestätigung	→	TOT:Reservierungsbestätigungskopf, TOT:Reservierungsbestätigungsposition	1	
TOT:Reservierungsbestät.kopf, TOT:X_kopf	→	KOT:Auftragskopf	1	2
TOT:Reservierungsbestät.pos., TOT:X_position	→	KOT:Auftragsposition	1	
TOT:Reservierungsbestät.kopf, TOT:X_kopf	→	KOT:Auftragskopf.Version2	2	
…	…	…		
VOT:Serviceanweisung>	→	VOT:Serviceanweisung>, IOT:MCKS_Serviceanweisung	2	
VOT:>Servicezusammenfassung	→	VOT:>Servicezusammenfassung IOT:MCKE_Servicezusammenfassung	2	
…	…	…		

Tabelle 6.21: Aktualisierte Konsolidierungsbeziehungen$_2$ (Szenario FIS)

Die konsolidierten Schemata KOS_2 und VOS_2 (Ergebnis A.P2) gehen als Input in die Transformationsspezifikation T2 ein. Sie werden in die initialen Schemata des REST-Architekturmodells überführt und lösen dort die erste Analyseaktivität D.A1 aus. Auf Basis des ermittelten Modell-Deltas$_{1,2}$, der konsolidierten Schemata von OS-ED$_1$, VOS-ED$_1$, Ressourcen$_1$ und Dokumenten$_1$ sowie der Konsolidierungsbeziehungen$_1$ werden die REST-Überarbeitungsschemata$_2$ erzeugt. Da bei der initialen Ableitung des REST-Architekturmodells und der Konsolidierung der initialen REST-Schemata keine tiefergehenden strukturellen Anpassungen erfolgen, wird auf eine explizite Darstellung verzichtet. Die zugehörigen Konsolidierungsbeziehungen$_1$ sind in den Tabellen des Anhangs A.5 aufgelistet. Nachfolgend werden die REST-Überarbeitungsschemata$_2$ in Abbildung 6.27 dargestellt.

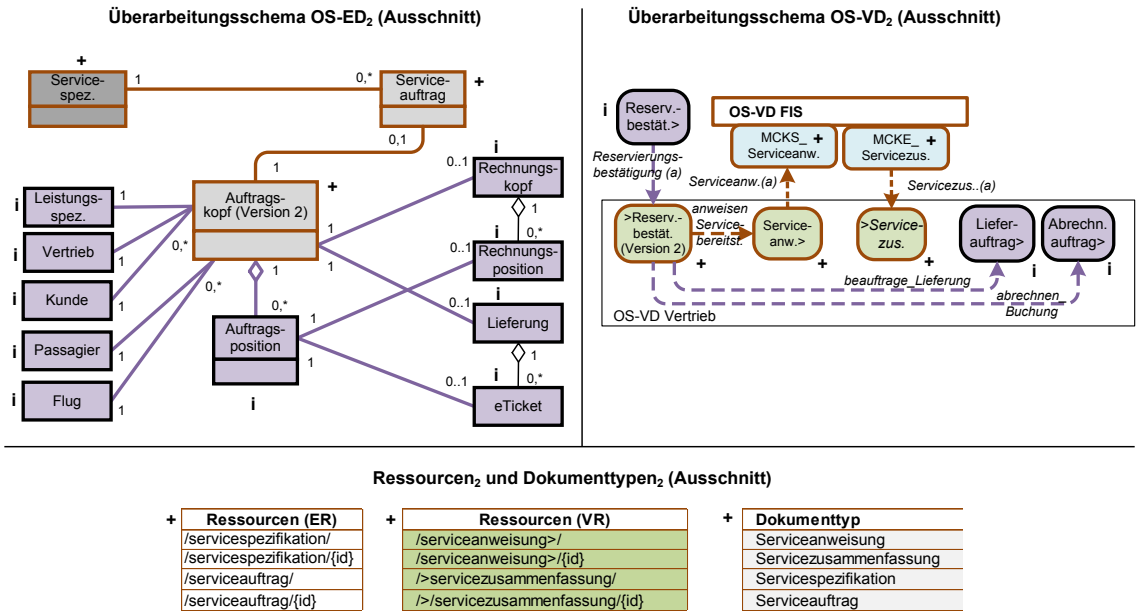


Abbildung 6.27: REST-Überarbeitungsschemata$_2$ (Szenario FIS)

Neben den hinzugefügten EOTs und eVOTs werden auch neue Ressourcenschnittstellen sowie diejenigen Dokumenttypen abgeleitet, die im Rahmen der Konsolidierung K2 zu überarbeiten sind. Aufgrund der Erstellung der neuen Versionen von *KOT:Auftragskopf.Version2* und *VOT:>Reservierungsbestätigung.Version2* werden die benachbarten unveränderten Objekttypen als instabil markiert. Entsprechend den Schritten zur Durchführung der ersten Pflegeaktivität (D.P1) werden für die REST-Überarbeitungsschemata Handlungsempfehlungen ermittelt (nach Abschnitt 6.5.2.3). Die Handlungsempfehlungen sehen die Prüfung der instabil markierten Objekttypen dahingehend vor, ob eine Anpassung ihrer Funktionalität hinsichtlich der neuen Version des *KOT:Auftragskopf* erforderlich ist. Die hinzugefügten Objekttypen (+ OT) sind nach den Schritten der Konsolidierung von K2 zu überarbeiten. Anschließend wird in der zweiten Pflegeaktivität (D.P2) das REST-Architekturmodell vom Systementwickler konsolidiert.

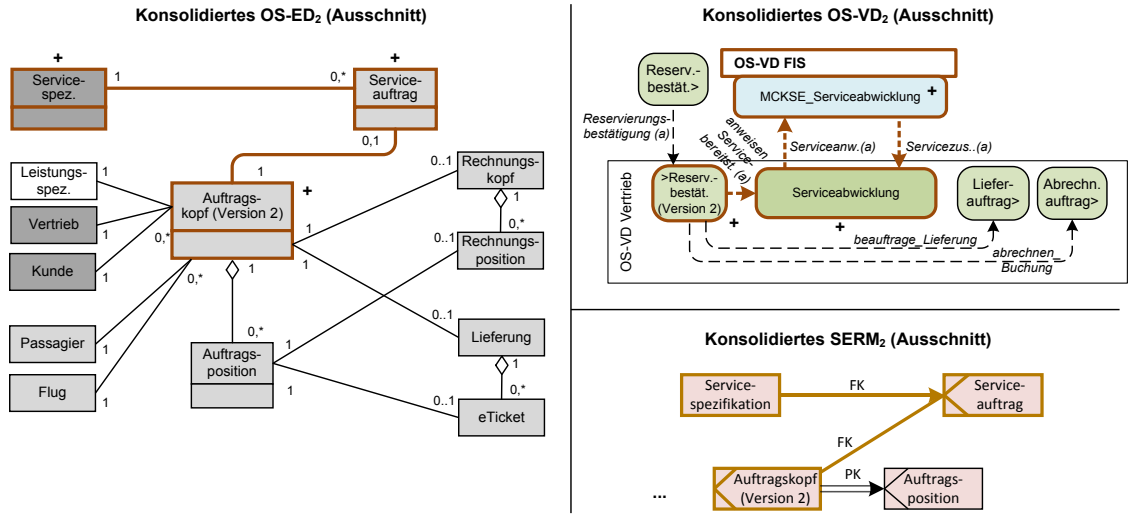


Abbildung 6.28: Konsolidiertes OS-ED$_2$ und Ausschnitt des konsolidierten OS-VD$_2$ (Szenario FIS)

Die konsolidierten Schemata$_2$ von OS-ED, OS-VD und SERM zeigt Abbildung 6.28. Die hinzugefügten Objekttypen werden im Rahmen der Konsolidierungsschritte von K2 überarbeitet und

um Spezifikationsdetails angereichert. Lediglich das Zusammenfassen der *eVOT:Serviceanweisung>* und *eVOT:>Servicezusammenfassung* zu einem *eVOT:Serviceabwicklung* sowie der zugeordneten R-IOTs werden hierbei als strukturelle Änderungen vorgenommen. Somit können die geänderten Bausteine der Schemata des REST-Architekturmodells in das Implementierungsmodell überführt werden und dort Maßnahmen zur Ablösung der alten Versionen festgelegt werden (vgl. Abschnitt 6.6.6). Neu zu erzeugen sind im Rahmen der Transformation T3 die Entitätsservices *Servicespezifikation*, *Serviceauftrag*, die neue Version von *Auftragskopf* sowie der Vorgangsservice *Serviceabwicklung* und die Funktionalität zur Kommunikation mit der Schnittstelle des externen Dienstleisters (R-IOT:MCKS/E_Serviceabwicklung). Zudem werden neue Dokumenttypen definiert und es wird eine Erweiterung der bestehenden Datenbasis um die Tabellen (DOT) zur Verwaltung der Servicedaten durchgeführt.

Nachdem die Behandlung der Änderungsoperation *Hinzufügen* beispielhaft für die Bausteine betriebliche Transaktion, betriebliches Objekt, Aufgabe sowie objektinternes Ereignis erläutert wurde, erfolgt im zweiten Schritt die Anwendung des Lösungsverfahrens der SOM-R-Methodik zur Systempflege für die Änderungsoperation *Entfernen*.

6.6.5.2 Zweites Szenario – Reduzierte Koordinierung des Versands

Gegenstand des zweiten Szenarios ist die Erhöhung der Autonomie bei der Durchführung der *Versandaufgabe* im Geschäftsprozess. So soll die Beauftragung eines Versendens von eTickets an einen Kunden vom Vertrieb nur noch angestoßen werden. Demnach entfällt eine weitere Kontrolle des Versandprozesses durch den Vertrieb. Die Version N wird für die anstehende Durchführung der Weiterentwicklung auf die Version 2 gesetzt (aktueller Stand nach dem ersten Szenario). Die Umsetzung dieser Änderung schlägt sich zunächst im IAS_2 durch das Entfernen der Transaktion *Lieferbestätigung* nieder. Abbildung 6.29 zeigt einen Ausschnitt von IAS und VES des angepassten SOM-Geschäftsprozessmodells$_3$ sowie der abgeleiteten initialen KOS_3 und VOS_3 (N+1 =3).

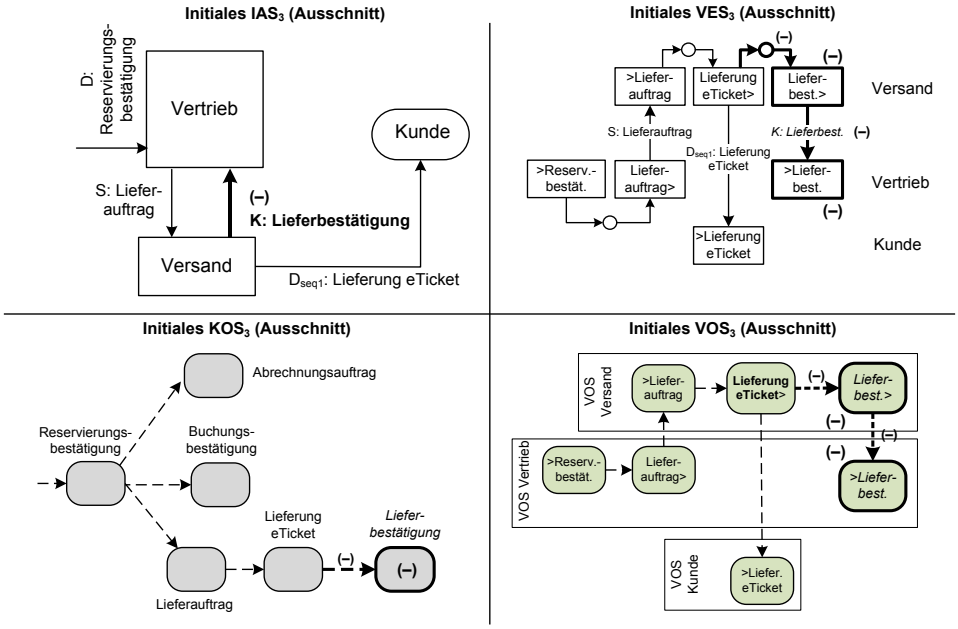


Abbildung 6.29: Dritte Version von SOM-GPM und fachlicher AwS-Spezifikation (Szenario Lieferbestätigung)

Als Änderung wird die betriebliche K-Transaktion *Lieferbestätigung* im IAS_2 entfernt. Diese beschreibt eine Kontrollmeldung zwischen Versand und Vertrieb, die von den Aufgaben *Lieferbestätigung>* und *>Lieferbestätigung* verarbeitet und gesendet bzw. empfangen wird. Die Abhängigkeit zwischen Transaktion und Aufgabe geht aus den im SOM-Metamodell definierten Beziehungen des Metaobjekts *Transaktion* zu weiteren Metaobjekten hervor. Die beiden korrespondierenden Aufgaben *Lieferbestätigung>* und *>Lieferbestätigung* der betrieblichen Objekte *Versand* und *Vertrieb* sowie die verknüpfenden objektinternen Ereignisse werden somit ebenfalls aus dem VES_2 entfernt. IAS_3 und VES_3 werden im Transformationsschritt T1 in das initiale KOS_3 und VOS_3 überführt. Die beiden Schemata stellen den Input der ersten Analyseaktivität (A.A1) dar.

Schema	Geändertes Modellobjekt
IAS (VES)	- bT:Lieferbestätigung(bO:Versand,bO:Vertrieb)
VES	- A:Lieferbestätigung>
VES	- A:>Lieferbestätigung
VES	- iE(A:>Lieferung_eTicket,A:Lieferbestätigung>)
iKOS	- TOT:Lieferbestätigung
iKOS	- interacts_with(TOT:Lieferung_eTicket,TOT:Lieferbestätigung)
iVOS	- VOT:Lieferbestätigung>(VOS:Versand)
iVOS	- VOT:>Lieferbestätigung(VOS:Vertrieb)
iVOS	- interacts_with(VOT:>Lieferung_eTicket,VOT:Lieferbestätigung>)
iVOS	- interacts_with(VOT:Lieferbestätigung>,>VOT:Lieferbestätigung)

Tabelle 6.22: Modell-$Delta_{2,3}$ von SOM-GPM und fachlicher AwS-Spezifikation (Szenario Lieferbestätigung)

Das in Schritt A.A1 ermittelte Modell-$Delta_{2,3}$ spezifiziert die Änderungen im strukturflexiblen SOM-GPM sowie in der initialen fachlichen AwS-Spezifikation. Tabelle 6.22 visualisiert das Delta in der vereinfachten Darstellungsform.

Die Ableitung der Überarbeitungsschemata erfolgt anschließend in der zweiten Analyseaktivität der Weiterentwicklung (A.A2). Die instabilen $Konsolidierungsbeziehungen_2$ von KOS und VOS zeigt Tabelle 6.23. Sie werden auf Basis der Verknüpfungen von geänderten zu unveränderten Bausteinen im Modell-Delta bestimmt. Beispielsweise setzte die entfernte *interacts_with-Beziehung* den unveränderten *TOT:Lieferung_eTicket* mit dem entfernten *TOT:Lieferbestätigung* in Beziehung. Die $Konsolidierungsbeziehungen_2$ zu den Objekttypen von *Lieferung_eTicket* sind folglich auch als instabil zu markieren.

	V	B	N
	…	…	…
(i)	TOT:Lieferung_eTicket	→	KOT:Flugticket
(i)	KOT:Lieferposition, KOT:Flugticket	→	KOT:eTicket
(i)	TOT:Lieferkopf, TOT:K:Lieferbestätigung	→	KOT:Lieferung
(i)	VOT:>Lieferung_eTicket	→	VOT:>Lieferung_eTicket
	…	…	…

Tabelle 6.23: Ausschnitt der markierten $Konsolidierungsbeziehungen_2$ (Szenario Lieferbestätigung)

Auf Grundlage des ermittelten Inputs (Modell-$Delta_{2,3}$, markierte $Konsolidierungsbeziehungen_2$ und konsolidierte $Schemata_2$) erfolgt die Bildung der $Überarbeitungsschemata_3$ von KOS und

VOS durch Anwendung des Lösungsverfahrens zu Schritt A.A2 (Abschnitt 6.6.2 und 6.6.3). Abbildung 6.30 visualisiert die Überarbeitungsschemata als Output der Aktivität A.A2.

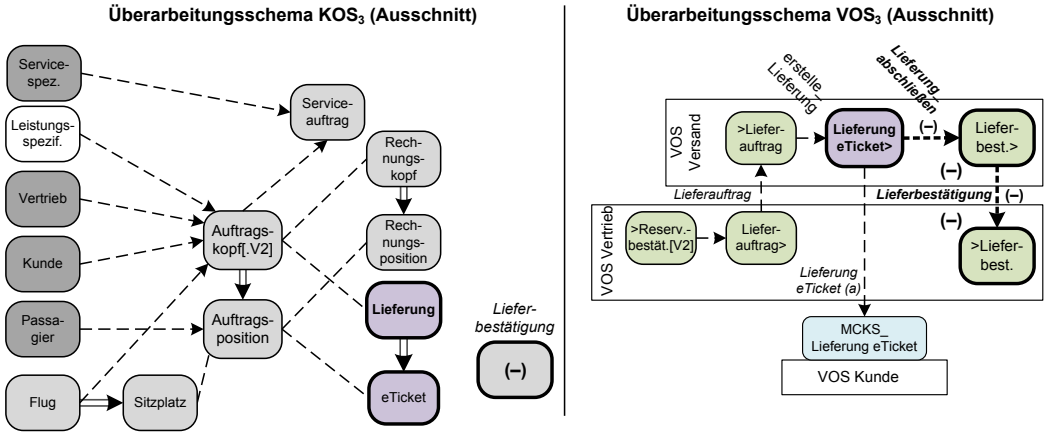


Abbildung 6.30: Überarbeitungsschemata$_3$ KOS und VOS (Szenario Lieferbestätigung)

Die TOTs *eTicket* und *Lieferung* werden im konsolidierten KOS$_2$ als instabil markiert, da sie bisher mit dem entfernten *TOT:Lieferbestätigung* des initiales KOS$_3$ zusammengefasst wurden und daher in Beziehung zu diesem standen. Das VOS-Überarbeitungsmodell$_3$ umfasst das konsolidierte VOS$_2$, in dem die beiden VOTs der *Lieferbestätigung* als entfernt markiert sind. Dem *VOT:Lieferung eTicket>* wird folglich eine Instabilitätsmarkierung zugewiesen.

Die Überarbeitungsschemata$_3$ von KOS und VOS bilden den Input für die Ermittlung von Handlungsempfehlungen in der ersten Pflegeaktivität (A.P1) (Abschnitt 6.5.1). Tabelle 6.24 zeigt die Handlungsempfehlungen für das vorliegende Beispiel.

Änderungsinstanz	Handlungsempfehlung
– KOT:Liefer-bestätigung	*Keine Überarbeitung*
	⇒ Prüfe das Anpassen von VOT-Lösungsverfahren in VOS:Vertrieb und VOS:Versand hinsichtlich (–) KOT:Lieferbestätigung
(i) KOT:Lieferung	*Anpassen der Eigenschaften* zur Realisierung der Interaktion (Empfang/Senden von +/-Nachrichten) und Berücksichtigung des geänderten – KOT:Lieferbestätigung
	⇒ Prüfe den verknüpften Objekttypen KOT:Auftragskopf
(i) KOT:eTicket	*Anpassen der Eigenschaften* zur Realisierung der Interaktion (Empfang/ Senden von +/-Nachrichten) und Berücksichtigung des geänderten – KOT:Lieferbestätigung
	⇒ Prüfe den verknüpften Objekttypen KOT:Auftragsposition
(i) VOT:Lieferung eTicket	*Anpassen der Eigenschaften* zur Realisierung der Interaktion (Empfang/ Senden von +/-Nachrichten) und Berücksichtigung des geänderten – VOT:Lieferbestätigung>
	⇒ Prüfe den verkn. VOT:>Lieferauftrag und IOT:MCKS_Lieferung_eTicket
– VOT:Liefer-bestätigung>	*Keine Überarbeitung*
	⇒ Prüfen von VOT-Lösungsverfahren bzgl. (–) VOT:Lieferbestätigung> ⇒ Prüfen von Eigenschaften des verknüpften (i) VOT:Lieferung eTicket ⇒ Prüfen der Konsolidierung abhängiger Objekttypen
– >VOT:Liefer-bestätigung	*Keine Überarbeitung*
	⇒ Prüfen von VOT-Lösungsverfahren bzgl. (–) VOT:>Lieferbestätigung ⇒ Prüfen der Konsolidierung abhängiger Objekttypen

– interact_with(Lieferung eTicket>,Lieferbest.>)	*Keine Überarbeitung*
	⇒ Prüfen von Eigenschaften des VOT:Lieferung eTicket, der mit der (–) Interaktionsbeziehung verknüpft war (Nachricht-Empfang/Senden). Prüfen von (–) VOT:Lieferbestätigung> entfällt, da bereits entfernt.
– interact_with (Lieferbest.>,>Lieferbest.)	*Keine Überarbeitung*
	⇒ Prüfen von Eigenschaften der (–) VOT:Lieferbestätigung> und (–) VOT:>Lieferbestätigung (entfällt, da bereits entfernt).

Tabelle 6.24: Ermittelte Handlungsempfehlungen$_3$ (Szenario Lieferbestätigung)

Beispielsweise sind die Eigenschaften des instabilen *VOT:Lieferung_eTicket>* hinsichtlich des Wegfalls der Interaktion mit *VOT:Lieferbestätigung>* zu prüfen und zu überarbeiten. Die „Platzhalter“ in den Auswirkungen der jeweils vorliegenden Änderungstypen werden anhand des KOS-Überarbeitungsschemas$_{N+1}$ konkretisiert. Die VOTs von VOS *Vertrieb* und *Versand* könnten auf den entfernten *TOT:Lieferbestätigung* bzw. die *KOT:Lieferung* und *KOT:eTicket* als Ergebnis der Konsolidierung zugreifen. Entsprechend sind deren Lösungsverfahren hinsichtlich einer Berücksichtigung dieser Änderung zu prüfen und ggf. anzupassen. Das Entfernen der VOTs (und interacts_with-Beziehungen) der *Lieferbestätigung* zieht primär ein Prüfen der Lösungsverfahren von VOTs nach sich, welche potenziell auf diese zugegriffen haben.

Die pflegende Konsolidierung des Anwendungsmodells führt der Systementwickler in der zweiten Pflegeaktivität (A.P2) durch (Abschnitt 6.6.4). Die bereitgestellten Assistenzinformationen unterstützen dabei die Weiterentwicklungstätigkeit. Einen Ausschnitt des konsolidierten KOS$_3$ und VOS$_3$ als Ergebnis der pflegenden Schema-Konsolidierung des Anwendungsmodells zeigt Abbildung 6.31.

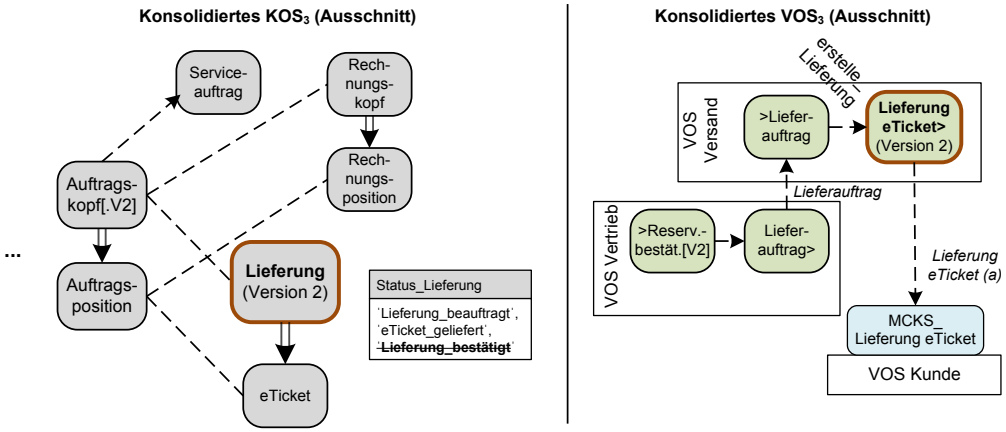


Abbildung 6.31: Konsolidiertes KOS$_3$ und konsolidiertes VOS$_3$ (Ausschnitt, Szenario Lieferbestätigung)

Bei der Spezifikation des konsolidierten KOS$_3$ wird der Wegfall der Eigenschaften von *TOT:Lieferbestätigung* bei der Erstellung der neuen Version von *KOT:Lieferung* berücksichtigt. Eine Überarbeitung von weiteren verknüpften KOTs ist dagegen nicht erforderlich, da bei den bestehenden Zugriffen auf die Attribute von *KOT:Lieferung* die Informationen der entfernten Bestätigung in der Fallstudie nicht durch andere Objekttypen abgefragt werden. Die Konsolidierung des VOS sieht die Überarbeitung des *VOT:>Lieferung_eTicket* vor. Der aktualisierte Stand der Konsolidierungsbeziehungen$_3$ wird parallel dazu dokumentiert (Tabelle 6.25).

V	B	N	A	I
...	...	...		
TOT:Lieferung_eTicket	→	KOT:Flugticket	1	
TOT:Lieferauftrag	→	KOT:Lieferkopf, KOT:Lieferposition	1	
KOT:Lieferposition, KOT:Flugticket	→	KOT:eTicket	1	
TOT:Lieferkopf, TOT:Lieferbestätigung	→	KOT:Lieferung	1	3
TOT:Lieferkopf	→	KOT:Lieferung[.Version2]	3	
VOT:>Lieferung_eTicket	→	VOT:>Lieferung_eTicket[.Version2]	3	
...	...	...		

Tabelle 6.25: Aktualisierte Konsolidierungsbeziehungen$_3$ von KOS und VOS (Szenario Lieferbestätigung)

Die konsolidierten Schemata KOS_3 und VOS_3 gehen als Input in die Transformationsspezifikation T2 ein und werden in die initialen Schemata des REST-Architekturmodells überführt. Nach Durchführung der Analyseaktivität D.A1 erfolgt die Erstellung der REST-Überarbeitungsschemata D.A2. Das Ergebnis des Entwicklungsschrittes fasst Abbildung 6.32 zusammen.

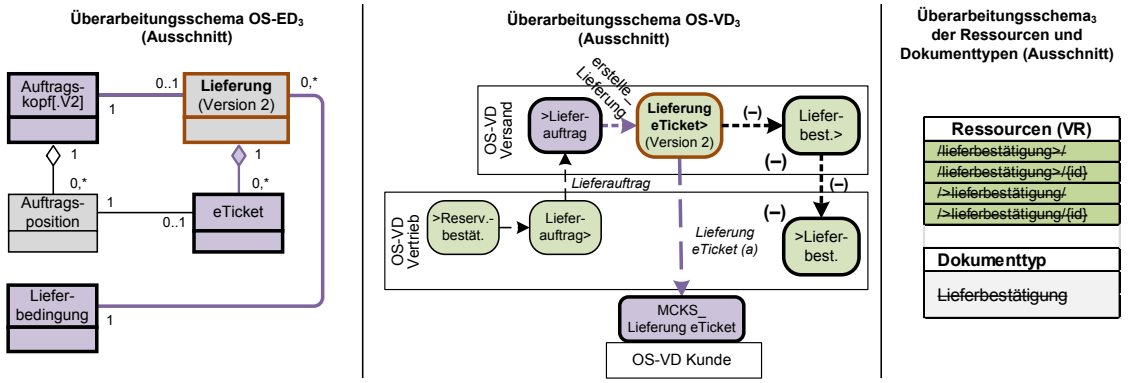


Abbildung 6.32: Überarbeitungsschemata$_3$ des REST-Architekturmodells (Szenario Lieferbestätigung)

Auf Basis des Überarbeitungsschemas werden Handlungsempfehlungen abgeleitet (D.P1). Aufgrund der Aktualisierung (Version 2) von *KOT:Lieferung* und *VOT:Lieferung eTicket>* sind beispielsweise die, mit diesen beiden Objekttypen, verknüpften Objekttypen als instabil zu markieren. Sie sind im Rahmen der Konsolidierung somit zu prüfen. Daneben sind die Lösungsverfahren aller Objekttypen zu prüfen, die auf die Ressourcen der aktualisierten sowie der entfernten Objekttypen zugreifen bzw. zugegriffen haben. Anschließend wird die Weiterentwicklung der Überarbeitungsschemata durchgeführt (D.P2).

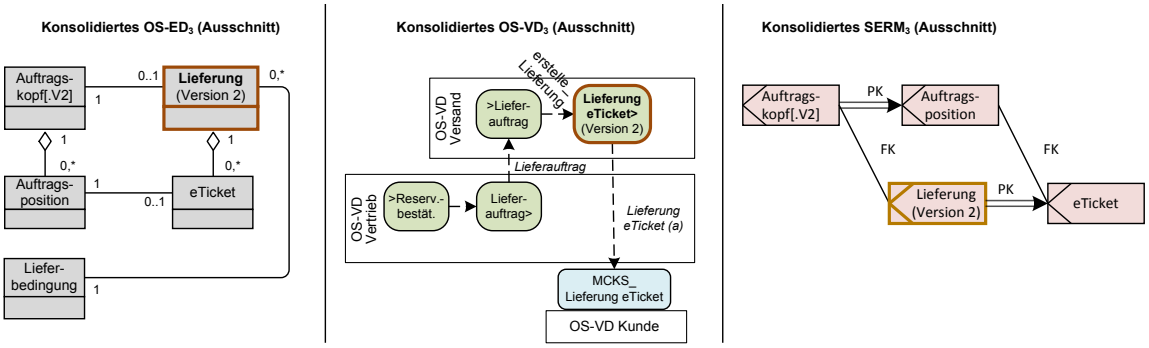


Abbildung 6.33: Konsolidierte Schemata OS-ED$_3$, OS-VD$_3$ und SERM$_3$ des REST-Architekturmodells (Szenario Lieferbestätigung)

Da in der Fallstudie keine weiteren strukturellen Überarbeitungen am Schema erfolgen, entsprechen die konsolidierten Schemata OS-ED_3, OS-VD_3 sowie Ressourcen- und Dokumentenschema$_3$ dem bereinigten Stand des jeweiligen Überarbeitungsschemas (Abbildung 6.33).

Die konsolidierten REST-Schemata$_3$ werden in die Spezifikationen des Implementierungsmodells überführt (Transformation T3). Auf der Implementierungsebene sind dabei insbesondere Maßnahmen zur Einführung der neuen Versionen bzw. Ablösung der alten Versionen von Entitätsdienst *Lieferung* und Vorgangsdienst *Lieferung eTicket>* festzulegen. Weitere Aufgaben sind die Stilllegung der veralteten Dienste, die bisher das Senden und den Empfang der *Lieferbestätigung* durchgeführt haben sowie die Evolution der Datenbasis. Im nachfolgenden Abschnitt 6.6.6 werden deshalb Anforderungen an die Weiterentwicklung von Diensten und der persistenten Datenhaltung in einem existierenden System betrachtet und damit abschließend die Bewältigung von Änderungen auf der Implementierungsebene kurz beleuchtet.

Anhand der Fallstudie Flugbuchung wird die Anwendung des konzipierten Lösungsverfahrens zur Systemweiterentwicklung demonstriert. Im ersten Szenario werden dem SOM-Geschäftsprozessmodell eine Reihe von GP-Bausteinen (bT, bO, A, iE) hinzugefügt und das bestehende Modellsystem schrittweise an die vorliegenden Änderungen angepasst. Anschließend werden in dem zweiten Szenario GP-Bausteine (bT, A, iE) aus dem Geschäftsprozessmodell entfernt und die Bewältigung der dazu korrespondierenden Änderungstypen gezeigt. Mit Ausnahme des Metaobjekts *Umweltereignis* werden somit alle Metaobjekte des SOM-Metamodells, die in dieser Arbeit relevant sind, exemplarisch untersucht. Das Ergebnis der Weiterentwicklung sind die angepassten Ebenen des Modellsystems, die wieder konsistent aufeinander abgestimmt sind. Die prototypische Implementierung eines Modellierungswerkzeugs zur Unterstützung der Durchführung der Systemweiterentwicklung mit der SOM-R-Methodik wird in Kapitel 7 vorgestellt.

6.6.6 Diskussion von Anforderungen an die Pflege des Implementierungsmodells

Die Überarbeitung von Anwendungs- und REST-Architekturmodell wird in der SOM-R-Methodik vollständig modellbasiert durchgeführt. Ergänzend zum Entwicklungsvorgehen der Methodik wird die Konsolidierung auf der Ebene des Implementierungsmodells zusätzlich durch die Aspekte der Realisierung des modellgetriebenen Entwicklungsvorgehens sowie der im Projekt angewandten Strategie des Managements von Änderungen beeinflusst (Abschnitte 5.5.6 und 5.5.7).

Bei der Weiterentwicklung eines existierenden, bereits implementierten Softwaresystems ergeben sich für die Implementierung somit weitere Anforderungen, welche die Durchführung dieser Aufgabe beeinflussen. Im Folgenden werden diesbezüglich Besonderheiten hervorgehoben und diskutiert.

MODELLGETRIEBENE WEITERENTWICKLUNG DER RESTFUL SOA

Die Generierung der Komponenten eines lauffähigen Softwaresystems RESTful SOA ist die Zielsetzung des letzten Transformationsschrittes T3 der SOM-R-Methodik. Wie bereits in Abschnitt 5.5.6 ausgeführt, könnte im Falle eines vollständigen Erreichens dieses Ziels die

Spezifikation des weiterentwickelten Systems durch *Neu-Generierung* erstellt werden. Somit ließe sich die Weiterentwicklung des Implementierungsmodells auf die Durchführung des dritten Transformationsschrittes reduzieren.

Häufig ist jedoch eine manuelle Ergänzung der generierten Artefakte erforderlich, um die Spezifikation des Implementierungsmodells abzuschließen. Typische Aufgaben sind dabei das Einfügen von individuellem Code in die Klassenmethoden und die Konfiguration der eingesetzten Basismaschinen (vgl. Schema-Konsolidierung K3, Abschnitt 5.5.7). Als Bestandteile einer RESTful SOA sind dabei insbesondere Entitätsdienste, (elementare/nicht-elementare) Vorgangsdienste sowie die Spezifikationen der Dienstschnittstellen und der persistenten Datenhaltung (Datenbankschema) zu prüfen, und ggf. manuelle Anpassungen und eine konsistente Abstimmung mit den geänderten Systemanforderungen vorzunehmen. Das Vorgehen entspricht der Durchführung der pflegenden Schema-Konsolidierung aus den vorangegangenen Abschnitten.

Evolution des eingeführten service-orientierten Systems

Den Gegenstand der Weiterentwicklung auf der Implementierungsebene bilden Systeme, die bereits in Betrieb genommen sind und damit im laufenden Geschäftsbetrieb eingesetzt werden. An die Durchführung der Systemweiterentwicklung schließt die Aufgabe des *Managements von Änderungen* auf der Ebene des implementierten Systems an, welche die Stilllegung veralteter und Inbetriebnahme neuer Systembestandteile sowie die Übernahme von Geschäftsdaten behandelt (z. B. [Somm12, S. 739 ff.], [Balz11, S. 529 f.]).

Im Fokus der SOM-R-Methodik steht die Weiterentwicklung der RESTful SOA. Hierzu werden die für die Durchführung des Entwicklungsprozesses erforderlichen Artefakte verwaltet. Die Untersuchung von Lösungen zur Umsetzung projektspezifischer Strategien für das Management von Komponentenversionen des implementierten Systems stellt dagegen kein Ziel der vorliegenden Arbeit dar. Zur Realisierung projektspezifischer Änderungsstrategien ist jedoch grundsätzlich eine Erweiterung von Vorgehen und/oder Hilfsmittel der SOM-R-Methodik denkbar. Hierzu sei auf die allgemeine Literatur zur Wartung und Pflege von Softwaresystemen sowie deren Unterstützung durch das Konfigurations- und Änderungsmanagement verwiesen (z. B. [Somm12, S. 739 ff.], [Balz11, S. 529 ff.], [PfAt10, S. 561 ff.], [Pres10, S. 584 ff.], [MeDe08]). Ausführungen zur Verwaltung der Änderungen in service-orientierten Systemen und der Versionierung ihrer Komponenten finden sich zudem in der weiterführenden SOA-Literatur (z. B. [ECP+13], [Josu08], [StTi07a], [KBS05]).

Nachfolgend werden zwei zentrale Problembereiche einer Verwaltung von geänderten Diensten und der Datenhaltungsebene in (RESTful) SOA hervorgehoben, um abschließend weitere Aufgaben sowie mögliche Lösungsansätze in diesem Aufgabenfeld zu skizzieren.

- *Bewältigung von Serviceänderungen.* Im Rahmen der Weiterentwicklung von RESTful SOA können Dienste grundsätzlich neu hinzugefügt, entfernt oder ihre Version aktualisiert werden. Für das Hinzufügen von neuen Diensten zu einer Systemversion werden Lösungsansätze benötigt, mit denen eine nahtlose Integration in die bestehende RESTful SOA realisierbar ist.

Von besonderem Interesse bei einer Bewältigung von Serviceänderungen ist das Entfernen eines veralteten Dienstes bzw. dessen Ablösen durch eine aktualisierte Version, da der Dienst unter Umständen weiterhin zugreifbar bleiben muss. Die Entwicklung einer geeigneten Strategie zur *Serviceablösung* und *Serviceversionierung* ist hierfür erforderlich (z. B. [ECP+13, S. 343 ff.], [Josu08, S. 169 ff.]).

Eine ausführliche Diskussion zum Lebenszyklus-Management und der Versionierung von Diensten führt JOSUTTIS [Josu08, S. 179 ff.]. Sowohl Entitäts- als auch Vorgangsdienste werden in einem laufenden Softwaresystem nach ihrer „Ablösung" normalerweise nicht aus der SOA entfernt. So sollten veraltete *Entitätsdienste* zugreifbar bleiben, solange es weitere Komponenten in der RESTful SOA gibt, die diese nutzen können [Josu08, S. 175 f.]. Das Ablösen von Vorgangsdiensten korrespondiert dabei mit dem Problem der Behandlung von Flexibilität zur Entwurfszeit und der Ausführungszeit in Workflow-Management-Systemen. Die hierbei eingesetzten Lösungen erscheinen auch für eine Anwendung auf den Bereich der Vorgangsdienste geeignet [DRR11].

Auch die REST-Prinzipien können bei der Versionierung von Diensten eine wichtige Rolle spielen, ermöglichen diese doch den Zugriff auf neue Dienste bzw. Dienstversionen über die einheitliche Schnittstelle. Ein wesentlicher Aspekt beim Austausch von Diensten oder ihrer Integration in ein bestehendes System ist somit die Kenntnis der Semantik von veröffentlichten Ressourcen der neu hinzugefügten Dienste.

- *Bewältigung von Schemaänderungen*. Veränderte Anforderungen an die RESTful SOA führen meist auch zur Anpassung der spezifizierten Datenbankschemata. Die Bewältigung von Schemaänderungen in laufenden Systemen erfordert die Bereitstellung von Lösungsansätzen zur Adaption persistenter Daten an eine aktualisierte Datenhaltungsstruktur.

  Zur Behandlung von Schemaänderungen werden in der Datenbankliteratur häufig Ansätze aus dem Bereich der *Schemaevolution*[64] diskutiert (z. B. [RaBe06]). Diese untersuchen die Überführung von persistent gespeicherten Dateneinheiten in geänderte Schemaversionen (sog. Instanzadaption) [BaGü09, S. 207]. Mit dem Ziel, die Informationen aus den ursprünglichen Versionen eines Datenbankschemas zu bewahren, werden zudem Konzepte zur *Schemaversionierung* untersucht ([BaGü09, S. 207], [RAL+06, S. 6]).

Mit den vorausgegangenen Ausführungen zu Problemen und Besonderheiten einer Weiterentwicklung des Implementierungsmodells wurden die Ränder des Betrachtungsgegenstandes der vorliegenden Arbeit noch einmal genauer ausleuchtet.

[64] Eine bibliographische Sammlung von Veröffentlichungen zum Thema Schemaevolution findet sich unter http://se-pubs.dbs.uni-leipzig.de/ (Abruf am 2. März 2015).

6.7 Reflexion der konzeptuellen Ergebnisse

Das Ziel des sechsten Kapitels ist die Erweiterung der SOM-R-Methodik um einen Ansatz zur Unterstützung der modellbasierten Anpassung von RESTful SOA an die Änderungen in strukturflexiblen SOM-Geschäftsprozessmodellen. Der erarbeitete Ansatz der Systempflege stellt den Lösungsbeitrag für das Konstruktionsproblem der vorliegenden Arbeit dar (Abschnitt 1.2). Er umfasst im Kern drei methodische Hilfsmittel und ein Vorgehensmodell, welches deren Zusammenspiel spezifiziert.

Nachfolgend werden die zentralen Erkenntnisse, die hinsichtlich der Auswirkungen von Änderungen im SOM-R-Modellsystem und ihrer Bewältigung, sowie allgemein der modellbasierten Pflege des SOM-R-Architekturmodells bestehen, verdichtet dargestellt (Abschnitt 6.7.1). Die Ausführungen schließen mit einer Prüfung der Ergebnisse bezüglich des Erreichens der Formalziele dieser Arbeit (Abschnitt 6.7.2).

6.7.1 Modellbasierte Überführung von Änderungen des SOM-Geschäftsprozessmodells in das REST-Architekturmodell

Das SOM-R-Architekturmodell und die Entwicklungsschritte der Modellbildung sind die methodische Basis für die Konzeption des Entwicklungsansatzes zur Systempflege. Durch die Bildung von struktur- und verhaltensorientierten Sichten auf die integrierten Metamodelle der Modellebenen wird ein einheitliches Begriffssystem formuliert. Zusammen mit der Definition von Beziehungen zwischen den Modellebenen unterstützt das Begriffssystem die konsistente Abstimmung von Schemata im gesamten spezifizierten Modellsystem. Ergänzend dazu ermöglicht die Definition von Transformations- und Konsolidierungsschritten die methodische Anleitung der (automatisierten und nicht-automatisierten) Modellbildung auf den Ebenen des Architekturmodells.

Das Ergebnis der modellbasierten Spezifikation von Informationssystemen mit der SOM-R-Methodik sind die Schemata auf der Ebene des REST-Architekturmodells. Deren Bestandteile besitzen klar abgegrenzte Verantwortlichkeiten im Anwendungssystem und fördern durch ihre Fokussierung auf die Semantik der Bausteine eine Entwicklung in einem modellbasierten Lösungsverfahren (Abschnitte 5.2, 5.4 und 5.3.2). Durch die Anwendung der SOM-R-Methodik werden RESTful Services (Innen- und Außensicht) entwickelt, deren existenzielle Abhängigkeiten durch einen oder mehrere Bausteine des SOM-Geschäftsprozessmodells durchgängig begründbar sind. Die Ressourcenorientierung der Zielarchitektur unterstützt damit die direkte Zuordnung von GP-Bausteinen zu einer abgegrenzten Menge von Ressourcen, ihrer Repräsentationen sowie der zugehörigen Objekttypen. Die Komponenten der RESTful SOA und ihre modellbasierte Spezifikation auf Basis eines SOM-GPM werden in Abschnitt 5.6 diskutiert.

Zusammenfassende Betrachtung der Auswirkungen von Änderungen in der Strukturansicht von SOM-Geschäftsprozessmodellen

Die Auswirkungen von Veränderungen in der Struktursicht (IAS) von SOM-Geschäftsprozessmodellen auf die RESTful SOA fasst Abbildung 6.34 zusammen. Abhängig von der durchgeführten GP-Änderungsoperation sowie den getroffenen Entwurfsentscheidungen im Anwendungsmodell lassen sich die Auswirkungen auf das REST-Architekturmodell in vier Fälle einteilen. Ein

wichtiges Kriterium zur Bestimmung möglicher Anpassungen ist dabei, ob bei der KOS-Konsolidierung ein Zusammenfassen von geänderten KOTs mit bereits existierenden (modellierten) KOTs vorgenommen wird.

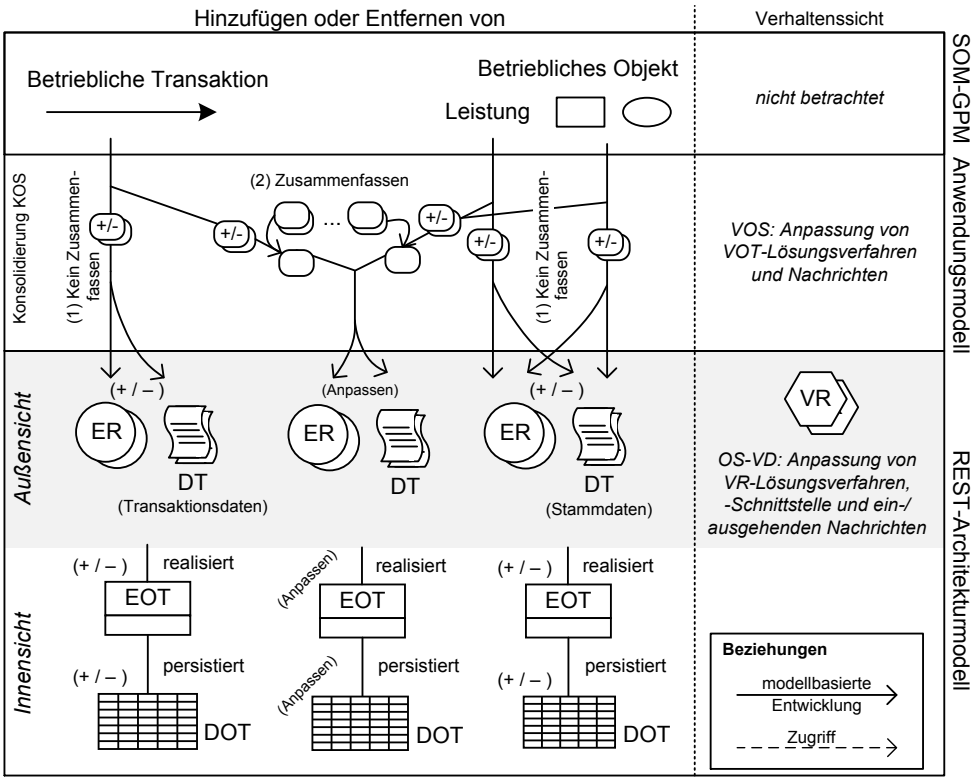


Abbildung 6.34: Auswirkungen von Änderungen im IAS auf das REST-Architekturmodell

Für den ersten Fall (1), dass im KOS *kein Zusammenfassen* von geänderten mit bestehenden Objekttypen durchgeführt wird, lassen sich die Auswirkungen einer GP-Änderung wie folgt beschreiben:

- Das *Hinzufügen* eines IAS-Bausteins resultiert auf der Ebene des REST-Architekturmodells im Hinzufügen einer Menge von Entitätsressourcen, ihren Repräsentationen (DTs) sowie zugehörigen EOTs und DOTs (Fall 1.1).
- Das *Entfernen* eines IAS-Bausteins zieht das Entfernen einer Menge von Entitätsressourcen, ihren Repräsentationen (DTs) sowie ein oder mehreren EOTs und DOTs nach sich (Fall 1.2).

In diesen Fällen sind die Änderungen von Bestandteilen im SOM-Geschäftsprozessmodell direkt auf ein Hinzufügen oder Entfernen von Komponenten der RESTful SOA hin überprüfbar.

Der zweite Fall (2) beschreibt das *Zusammenfassen* von geänderten KOTs mit unveränderten KOTs im Rahmen der KOS-Konsolidierung. Die bereits bestehenden Komponenten sind dann entsprechend der geänderten Funktionalität anzupassen:

- Das *Hinzufügen* eines IAS-Bausteins führt im REST-Architekturmodell zur Anpassung einer Menge von Entitätsressourcen, ihren Repräsentationen (DTs) sowie zugehörigen EOTs und DOTs (Fall 2.1).

- Das *Entfernen* eines IAS-Bausteins führt (ebenfalls) zur Anpassung einer Menge von Entitätsressourcen, ihren Repräsentationen (DTs) sowie einen oder mehreren EOTs und DOTs (Fall 2.2).

Die Komponenten der RESTful SOA sind in ihrer Existenz somit nicht nur von den geänderten, sondern auch von unveränderten Bestandteilen des SOM-Geschäftsprozessmodells abhängig. Alle Konsolidierungsentscheidungen des Anwendungsmodells und des REST-Architekturmodells, die unter Einbeziehung der geänderten Bausteine getroffen wurden, sind gesondert zu berücksichtigen. In diesen Fällen sind die Auswirkungen von Änderungen auf folgende Entwicklungsinformationen zu prüfen:

- Entwurfsentscheidungen und spezifizierte Eigenschaften der unveränderten Komponenten der RESTful SOA
- Workflows von Vorgangsressourcen und Anfrage-/Antwort-Nachrichten der REST-Schnittstellen

Die Änderungen des SOM-GPM sind somit in einem modellbasierten Vorgehen in die Änderungen einer eindeutig bestimmten Menge von Ressourcen (RESTful Service) überführbar. Die Anpassung des Gesamtsystems erfordert das Prüfen und Aktualisieren von Lösungsverfahren der, mit geänderten Bausteinen, verknüpften und unveränderten Bausteine. Die Ressourcenkoordination erfolgt dabei über die einheitliche Schnittstelle, deren Nutzung wesentlich durch die fachliche Bedeutung der entwickelten Komponente bestimmt wird.

Zusammenfassende Betrachtung der Auswirkungen von Änderungen in der Verhaltenssicht von SOM-Geschäftsprozessmodellen

Die Bestandteile, die in der Verhaltenssicht von SOM-Geschäftsprozessmodellen geändert werden können, sind Aufgaben, die durch objektinterne Ereignisse und betriebliche Transaktionen verknüpft werden. Abbildung 6.35 zeigt die Auswirkungen von Strukturänderungen im Vorgangs-Ereignis-Schema (VES) auf das REST-Architekturmodell.

Entsprechend den Erläuterungen zum IAS ist auch die Änderung einer Aufgabe im VES direkt auf eine korrespondierende Änderung von Vorgangsressourcen abbildbar, sofern im VOS kein Zusammenfassen mit weiteren (bereits existierenden) VOTs vorgenommen wird. Bei der Gestaltung der Service-Koordination besteht zudem auf der Ebene des OS-VD die Möglichkeit Bausteine zusammenzufassen. Existieren jedoch Abhängigkeiten zwischen einem geänderten Baustein und den dokumentierten Entwurfsentscheidungen auf einer Modellebene, so ist ein Prüfen und Anpassen der Ressourcen und ihrer Objekttypen erforderlich.

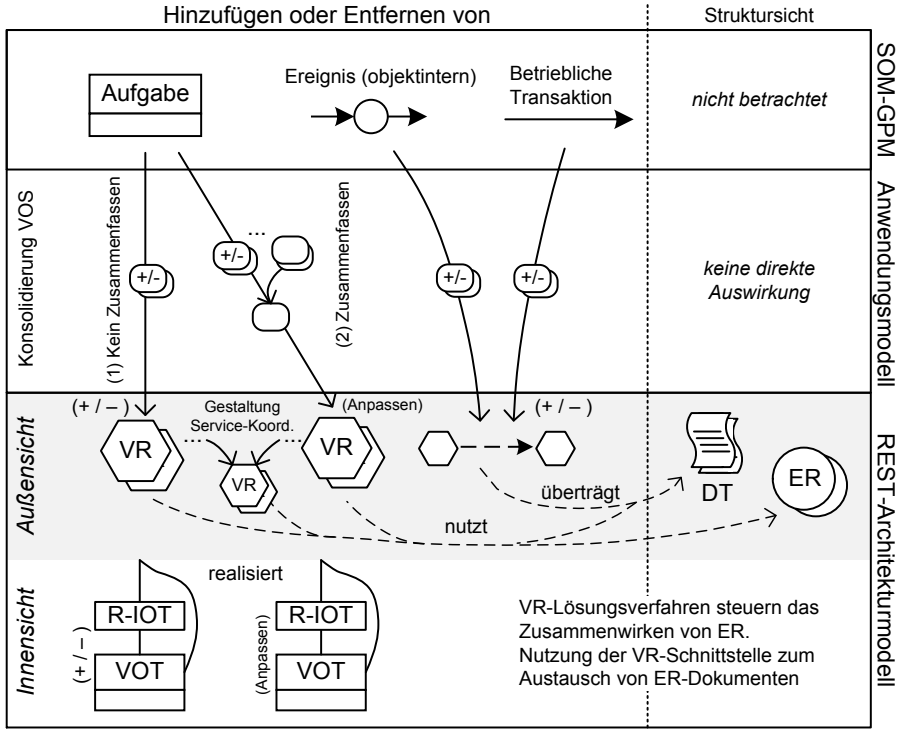


Abbildung 6.35: Auswirkungen von Änderungen im VES auf das REST-Architekturmodell

Grundsätzlich lassen sich drei Fälle von Auswirkungen einer GP-Änderung im VES abgrenzen:

- Das Hinzufügen oder Entfernen einer *Aufgabe* führt zum Hinzufügen bzw. Entfernen einer Menge von Vorgangsressourcen sowie der zugehörigen VOTs (e/ne) und R-IOTs, wenn bei der Konsolidierung *kein Zusammenfassen* erfolgt (Fall 1).
- Das Hinzufügen oder Entfernen einer *Aufgabe* resultiert in der Prüfung und Anpassung einer bestimmbaren Menge von Vorgangsressourcen sowie gebildeten VOTs (e/ne) und R-IOTs, wenn ein *Zusammenfassen* mit VOTs im VOS bzw. OS-VD erfolgt (Fall 2).
- Das Hinzufügen oder Entfernen eines *objektinternen Ereignisses* oder einer *betrieblichen Transaktion* führt zum Hinzufügen bzw. Entfernen einer interacts_with-Beziehung zwischen den Vorgangsressourcen zweier Dienste (Fall 3).

In allen drei Fällen bestehen keine direkten Auswirkungen auf die strukturorientierten Schemata des REST-Architekturmodels. Die Koordination zwischen bereits existierenden und hinzugefügten Vorgangsressourcen wird durch die Realisierung von Operatoren für den Aufruf der einheitlichen Schnittstelle und die Definition der dabei auszutauschenden Nachrichten realisiert. Unabhängig davon erfolgt ein Prüfen von Entwurfsentscheidungen sowie der spezifizierten Eigenschaften von unveränderten Komponenten der RESTful SOA.

Assistenzinformationen zur Unterstützung der modellbasierten Systempflege

Die modellbasierte Pflege von Informationssystemen ist Aufgabe des Systementwicklers. Eine große Herausforderung bei der zielgerichteten Anpassung eines existierenden SOM-R-Modellsystems ist die Bewältigung der mit der Durchführung dieser Aufgabe einhergehenden Komplexität. Als Ursachen dieser Komplexität sind insbesondere die verschiedenartigen Abhängigkeiten zwischen Bausteinen im spezifizierten Modellsystem, die semantische Lücke zwischen

den Ebenen von Quell- und Zielsystem sowie die Freiheitsgrade, die bezüglich der Durchführung der Modellbildung bestehen, zu nennen. Einen ersten Schritt zur Bewältigung dieser Herausforderung bildet die Definition eines methodischen Rahmens von Architektur- und Vorgehensmodell sowie von Entwicklungsschritten in der SOM-R-Methodik.

Im zweiten Schritt wird eine Erweiterung der SOM-R-Methodik konzipiert, auf deren Basis der Systementwickler durch die Bereitstellung von Assistenzinformationen bei der Durchführung der Weiterentwicklungsaufgabe unterstützt wird. Als wichtige Gründe dafür, dass die Weiterentwicklung der RESTful SOA ebenfalls durch den vorgeschlagenen modellbasierten Ansatz behandelt werden kann, sind insbesondere zwei Punkte anzuführen:

- Die konzeptuelle Modellierung von RESTful SOA wird wesentlich durch die Merkmale der Ressourcen- und Dokumentenorientierung unterstützt, welche die Abhebung des Abstraktionsgrades der softwaretechnischen Modellebene ermöglichen. Im Zentrum der Entwicklung steht somit die Spezifikation von Ressourcen der REST-Modellebene und weniger die Definition von anwendungsspezifischen Schnittstellen.
- Die Weiterentwicklung der RESTful SOA wird zudem durch die klare Trennung von Verantwortlichkeiten einzelner Modellbausteine und der definierten Baustein-Beziehungen gefördert. Die Bedeutung der Bausteine des SOM-Geschäftsprozessmodells bleibt in den Bausteinen auf der REST-Modellebene erhalten. Dies fördert eine enge Bindung der beiden Modellebenen.

Die Pflege des SOM-R-Modellsystems ist mit dem vorgeschlagenen Lösungsverfahrens durchgängig modellbasiert realisierbar. Zur Unterstützung der Identifikation und Analyse von Änderungen, und der pflegenden Konsolidierung einer Modellebene werden folgende Informationen ermittelt:

- Die Modelldifferenz zwischen der vorherigen und der (neuen) aktuellen Schemaversion wird in Form eines Modell-Deltas dargestellt.
- Die dokumentierten Entwurfsentscheidungen, die bei der Überarbeitung der vorausgegangenen (ursprünglichen) Schemaversionen getroffen wurden. Entscheidungen, die von Schemaänderungen betroffen sind, werden hervorgehoben.
- Das Überarbeitungsmodell, welches eine Überführung und Integration des Modell-Deltas der aktuellen Version eines initialen Schemas in die ursprüngliche Version des zugehörigen konsolidierten Schemas darstellt.
- Eine Menge von Handlungsempfehlungen, die ausgehend von den Änderungsmarkierungen im Überarbeitungsschema abgeleitet werden und dem Entwickler Hinweise zur Behandlung der vorliegenden Änderungen geben.

Die Plausibilität und Anwendung des vorgeschlagenen Lösungsverfahrens wird in Abschnitt 6.6.5 anhand des Beispiels der Fallstudie erläutert. Im Rahmen der prototypischen Entwicklung eines Modellierungswerkzeugs wird in Kapitel 7 die Implementierbarkeit einer maschinellen Bereitstellung von Assistenzinformationen demonstriert.

Die Ausführungen zur Konzeption des modellbasierten Ansatzes der Systempflege belegen das Erreichen des zweiten Untersuchungsziels. Damit kann auch das vollständige Untersuchungsziel der Arbeit als erfüllt beurteilt werden. Zusammenfassend ist festzuhalten, dass die methodischen Ergebnisse, die mit der SOM-R-Methodik erarbeitet werden konnten, die eingangs formulierten Fragen der Motivation beantworten (Abschnitt 1.1).

6.7.2 Prüfung der Formalziele

Im Folgenden wird die Einhaltung der Formalziele geprüft, die dem Untersuchungsziel der modellbasierten Systempflege an die Seite gestellt wurden (Abschnitt 1.2.2):

- Die Konzepte zur *Bewältigung der Komplexität* der Systemweiterentwicklungsaufgabe fußen in dieser Arbeit auf den Bestandteilen, die die SOM-R-Methodik für eine integrierte und durchgängige Modellbildung bereitstellt (vgl. Diskussion in Abschnitt 5.6). Im Fokus steht dabei die Behandlung von Änderungen durch die zielgerichtete Anpassung des (strukturflexiblen) Modellsystems. Die Komplexitätsbewältigung bei der Durchführung der Systementwicklung unterstützen folgende Hilfsmittel: (1) Die Identifikation von Schema-Änderungen erfolgt mit dem Modellvergleichsansatz in automatisierter Form. Er unterstützt damit eine Analyse der Auswirkungen von Strukturflexibilität (Abschnitt 6.4). (2) Als zentrales Artefakt der Weiterentwicklung dient das Überarbeitungsschema, welches dem Systementwickler zur Verfügung gestellt wird (Abschnitt 6.6.3). Es integriert die Informationen über identifizierte Schema-Änderungen mit den Entwurfsentscheidungen der vorhergehenden Entwicklung. (3) Die Durchführung der nicht-automatisierten Aufgabe der Pflege von Schemata des geänderten Modellsystems unterstützt ein Empfehlungsansatz (Abschnitt 6.5). (4) Der systematische Ablauf des Entwicklungsprozesses kann durch das Vorgehensmodell der Systempflege sichergestellt werden (Abschnitt 6.6.2).

- Die *Sicherung der Konsistenz* in der angepassten Spezifikation des Modellsystems fördert die SOM-R-Methodik im Wesentlichen durch den Einsatz der methodischen Hilfsmittel, die bereits in Abschnitt 5.6.2 erläutert wurden. Hierunter fallen vor allem die Bereitstellung von integrierten Metamodellen und Metaphern auf den Modellebenen, von Beziehungsmetamodellen sowie den definierten Transformationen und Konsolidierungen zur Anleitung der Modellerstellung. Das Prüfen einer konsistenten Ausrichtung der RESTful SOA auf das SOM-GPM wird zudem durch den Dokumentationsansatz unterstützt (Abschnitt 6.3). Durch seinen Einsatz sind die Beziehungen zwischen den gebildeten Modellelementen innerhalb und zwischen den Ebenen des Modellsystems transparent und durchgängig dokumentiert und die Entwurfsentscheidungen der Entwicklung nachvollziehbar. Das Defizit, welches bezüglich der Herstellung von Nachvollziehbarkeit bisher in der SOM-R-Methodik bestand, wird durch den Dokumentationsansatz beseitigt (siehe Abschnitt 5.6.2).

- Der „modulare“ Aufbau der konzipierten methodischen Hilfsmittel stützt das Erreichen des Ziels der *Flexibilität* der Entwicklungsmethodik. Die einzelnen Bestandteile des Lösungsansatzes zur Systempflege sind dadurch an geänderte Anforderungen der Systemweiterentwicklungsaufgabe anpassbar. Beispielsweise sind eine Detaillierung

der dokumentierten Entwurfsschritte, eine Änderung des Detailgrades der Modelldifferenzen, oder auch eine Erweiterung der Empfehlungsbasis denkbar. Da diese Bestandteile in die Bestimmung des Überarbeitungsschemas einfließen und ihre Integration durch das Lösungsverfahren der Systempflege erfolgt, muss hierbei stets auf eine konsistente Abstimmung der Anpassungen untereinander geachtet werden.

Abschließend ist festzuhalten, dass auch die Formalziele dieser Arbeit durch den konstruierten Lösungsansatz erfüllt werden (Abschnitt 1.2.2). Offen ist jedoch weiterhin die Bereitstellung eines Softwarewerkzeugs als Realisierung einer maschinellen Unterstützung der gesamten Systementwicklungsaufgabe. Die Konzeption und prototypische Implementierung eines Entwicklungswerkzeugs behandelt Kapitel 7.

6.8 Zusammenfassung

Gegenstand des sechsten Kapitels ist die Konstruktion eines Ansatzes zur modellbasierten Pflege betrieblicher Informationssysteme. Die Weiterentwicklung eines bestehenden Informationssystems wird von veränderten Anforderungen im strukturflexiblen SOM-Geschäftsprozessmodell ausgelöst. Das Anwendungssystem des IS ist als RESTful SOA realisiert. Im Modellsystem manifestieren sich Veränderungen in Form eines Hinzufügens oder Entfernens von Modellbausteinen und den Beziehungen zwischen diesen Bausteinen.

Ausgehend von den bestehenden methodischen Rahmenbedingungen sowie dem Strukturmodell der Entwicklungsaufgabe werden zunächst die Anforderungen an eine Erweiterung der SOM-R-Methodik abgeleitet. Auf Basis dieser Anforderungen wird ein Lösungsansatz zur modellbasierten Pflege betrieblicher Informationssysteme konstruiert. Den Kern der erweiterten Methodik bilden Hilfsmittel, deren Ziel das Erzeugen von Assistenzinformationen ist. Diese Informationen sollen den Systementwickler bei der Analyse von Veränderungen, der Durchführung von Systemanpassungen und der konsistenten Abstimmung der Ebenen im gebildeten Modellsystem unterstützen. Das Vorgehen zur Durchführung der modellbasierten Weiterentwicklung sieht die schrittweise Behandlung von Strukturänderungen auf den Modellebenen des SOM-R-Architekturmodells korrespondierend zur Modellbildung mit der SOM-R-Methodik (Kapitel 5) vor.

Das erarbeitete Lösungsverfahren zur Systemweiterentwicklung basiert auf dem Einsatz von drei Hilfsmitteln. Ziel des *Dokumentationsansatzes* ist das Erfassen von strukturellen Entwurfsentscheidungen und Entwicklungsartefakten, die im Rahmen der Systementwicklung auf den Ebenen des SOM-R-Architekturmodells getroffen bzw. erstellt werden. Dieser Ansatz unterstützt die Verfolgbarkeit zwischen Bausteinen in SOM-Geschäftsprozessmodellen und den Komponenten der RESTful SOA durch die durchgängige Dokumentation ihrer Abhängigkeiten. Er fördert damit die konsistente Abstimmung der Modellebenen. Mit der Konzeption des *Modellvergleichsansatzes* wird die Bestimmung der Modelldifferenz zwischen zwei Versionen eines Schemas angestrebt. Eine Differenz wird in einem Modell-Delta spezifiziert. Die Unterschiede zweier Schemata werden auf Basis des sprechenden Bezeichners der Schemaobjekte (Vergleichskriterium) sowie der Bestimmung von elementaren Änderungsoperationen (Hinzufügen/ Entfernen) ermittelt. Diese beiden Einschränkungen bezüglich der Identifikation

und Beschreibung von Änderungen dienen der Reduzierung der Komplexität und Abstimmung des betrachteten Modelldetailgrades mit den Anforderungen der erweiterten SOM-R-Methodik. Die dokumentierten Entwicklungsinformationen und das Modell-Delta fließen als Input in die Bildung eines *Überarbeitungsschemas* ein. Dieses stellt das zentrale Artefakt zur Pflege einer Modellebene dar. Das Überarbeitungsschema integriert das Modell-Delta, die aktuelle sowie die ursprünglich entwickelte Version einer Modellebene bzw. deren Schemata. Auf Grundlage des Überarbeitungsschemas kann die Ableitung von Maßnahmen zur Behandlung bestehender Änderungen durch den *Empfehlungsansatz* erfolgen, dessen Empfehlungsbasis in einer initialen Version vorgestellt wird. Die schrittweise Durchführung des Lösungsverfahrens der Systempflege und die Nutzung der methodischen Hilfsmittel werden anhand einer detaillierten Darstellung des *Vorgehensmodells* und dem Zusammenspiel seiner Bestandteile erläutert. Das Vorgehensmodell spezifiziert das Lösungsverfahren zur Durchführung der Systemweiterentwicklungsaufgabe.

Das Überarbeitungsschema, die dokumentierten Entwicklungsinformationen sowie die Handlungsempfehlungen bilden die Assistenzinformationen, die dem Entwickler zur Durchführung von Systemanpassungen bereitgestellt werden. Der Prozess der Weiterentwicklung wird somit durch einen vollständig modellbasierten Ansatz unterstützt. Die Plausibilität und Anwendung des erarbeiteten Lösungsverfahrens wird anhand zweier Szenarien der eingeführten Fallstudie gezeigt. Die Ergebnisse dieses Kapitels werden mit Fokus auf die Auswirkungen von Änderungen im strukturflexiblen SOM-Geschäftsprozessmodell und deren Behandlung auf der Ebene des REST-Architekturmodells in einem Fazit dargestellt und reflektiert.

7 Konzeption und Realisierung eines Werkzeugprototyps

Gegenstand des siebten Kapitels ist die Konzeption und Implementierung eines Werkzeugprototyps als Bestandteil der konstruierten Entwicklungsmethodik. Zunächst werden die Entwicklungsplattform des Prototyps sowie die darin eingesetzten Technologien vorgestellt und die Rahmenbedingungen seiner Realisierung beleuchtet. Anschließend erfolgen die Erläuterung der einzelnen Komponenten des Werkzeugs und die Demonstration der Implementierbarkeit ihrer Funktionalität am Beispiel der Fallstudie. Die exemplarische werkzeuggestützte Anwendung der Methodik dient als weiterer Schritt zur Erbringung des Proof of Concepts der vorliegenden Arbeit.

7.1 Methodische Rahmenbedingungen und Wahl der Technologien

Die prototypische Implementierung des Entwicklungswerkzeugs der SOM-R-Methodik erfolgt auf Basis der Plattform *Eclipse* und ihrer Erweiterung durch die Technologien des *Eclipse Modeling Projects* (EMP)[65]. Den Kern der Modellbildung mit Eclipse bildet das *Eclipse Modeling Framework* (EMF)[66], das eine Plattform zur Erstellung (und Nutzung) abstrakter Modellierungssprachen und zur modellbasierten Codegenerierung ist. Die Spezifikation von domänenspezifischen Metamodellen und ihren Modellen basiert in EMF auf dem (Meta-)Metamodell *Ecore* [SBP+08, S. 17 f.].

Auf Basis von EMF ist eine Vielzahl weiterer Projekte wie z. B. Technologien zur Durchführung von Modelltransformationen (QVT, ATLAS) und Modellvergleichen (Compare), ein Model-Repository (CDO) sowie Frameworks zur Erstellung grafischer (GMF) oder textueller (TMF) Modellierungssprachen realisiert. Für nähere Informationen zu den einzelnen Projekten sei auf die EMP-Webseite[65] und die weiterführende Literatur verwiesen (z. B. [Gron09], [SBP+08]).
Die Eignung der EMF-Plattform und ihrer Technologien für die modellgetriebene Systementwicklung im Rahmen der SOM-Methodik wurde am Beispiel der Transformation von SOM-Anwendungsspezifikationen in die Softwareartefakte einer Komponentenarchitektur bereits untersucht und bestätigt [Häre14].

7.1.1 Modellbildung auf Basis von EMF

Die zentrale Voraussetzung für die Entwicklung des Werkzeugprototyps ist die Überführung der integrierten Metamodelle des SOM-R-Architekturmodells in EMF-Metamodelle. Modelle sowie Metamodelle werden in EMF auf Basis des Ecore-(Meta-)Metamodells erstellt, welches sich selbst als EMF-Modell definiert [SBP+08, S. 17 f.]. Den Kern des Meta-Metamodells von

[65] Offizielle Webseite: http://projects.eclipse.org/projects/modeling (Abruf am 24. Februar 2015).

[66] Offizielle Webseite: http://www.eclipse.org/modeling/emf/ (Abruf am 24. Februar 2015).

Ecore zeigt Abbildung 7.1. Das vollständige Meta-Metamodell wird aufgrund seines Umfangs nicht betrachtet. Es ist auf den Webseiten des EMP[67] dargestellt.

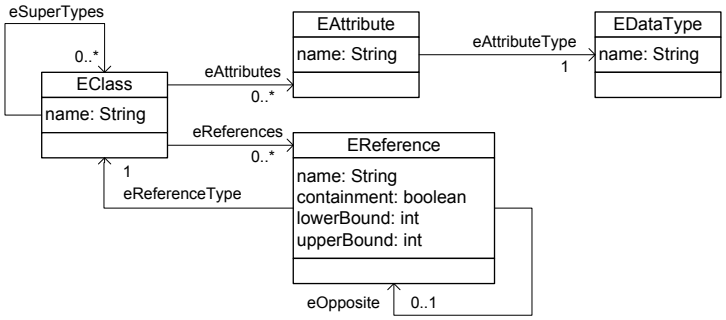


Abbildung 7.1: Kern des Ecore-Metamodells (in Anlehnung an [SBP+08, S. 105])

Die folgenden vier EClasses bilden die Kernbausteine von Ecore (nach [SBP+08, S. 18]):

- Die *EClass* repräsentiert einen Modellbaustein. Sie hat einen Namen und kann Referenzen auf beliebig viele EClasses sowie beliebig viele Attribute besitzen.
- Das *EAttribute* repräsentiert Attribute von EClasses. Jedes Attribut besitzt einen Typ.
- Die *EReference* beschreibt die Beziehung zwischen zwei Bausteinen. Jede Beziehung besitzt eine (min,max)-Kardinalität und kann EClasses enthalten (containment).
- Der *EDataType* repräsentiert elementare oder komplexe Datentypen von Attributen.

Die Metamodelle der SOM-R-Methodik basieren auf dem Meta-Metamodell von SINZ [Sinz96, S. 129] (Abschnitt 2.1.5). Die in dieser Arbeit angewandte Abbildung der Kernbausteine des Meta-Metamodells von Sinz auf das Ecore-(Meta)-Metamodell zeigt Tabelle 7.1.

Meta-Metamodell (Sinz)	Ecore-Metamodell
Metaobjekt	EClass
connects-Beziehung	EReference
is_a-Beziehung	Inheritance (eSuperType)

Tabelle 7.1: Beziehung zwischen dem Meta-Metamodell nach Sinz und dem Ecore-Metamodell

Jedes *Metaobjekt* des Meta-Metamodells (nach Sinz) wird in eine *EClass* überführt. *Generalisierungen* beschreiben Supertypen und damit Inheritance-Beziehungen zwischen EClasses. Die Standard-*Kardinalitäten* ((0,1), (1,1), (0,*), (1,*)) jeder connects-Beziehung werden in die Eigenschaften *Lower Bound* und *Upper Bound* der EReference überführt. Die (2,2)-Kardinalität einer Beziehung wird in Form zweier (1,1)-EReferences gebildet, um die Richtung der Beziehung darzustellen. EReferences, die im Ecore-Metamodell eine has-Beziehung oder eine connects-Beziehung im Sinne einer Aggregation modellieren, beschreiben ein *Containment*. Zudem wird in Ecore eine EClass explizit als Einstiegspunkt in das Schema definiert, welche sich aus den modellierten Schemabausteinen zusammensetzt (EReference mit Containment). Die Zuordnung weiterer Attribute oder die Definition eigener Datentypen ist möglich.

[67] Webseite des Ecore-Metamodells (Stand August 2014): http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html (Abruf am 24. Februar 2015)

7.1.2 Spezifikation der Modelltransformationen

Die Unterstützung einer Durchführung der modellbasierten Gestaltung und Pflege von Informationssystemen mit der SOM-R-Methodik erfordert den Einsatz von Technologien zur Spezifikation der Transformationsschritte (T1-T3) sowie der Erzeugung des Überarbeitungsschemas (A2) und die Ermittlung von Handlungsempfehlungen (P1). Die automatisierte Erzeugung von Modellen wird im Werkzeugprototyp dieser Arbeit auf Basis einer Transformationssprache implementiert.

Grundsätzlich ist zwischen den Transformationsarten *Model-to-Model* (M2M) sowie *Model-to-Text* (M2T) zu unterscheiden (z. B. Abschnitt 2.3.1, [Gron09, S. 277 f.]). Zur Unterstützung des Entwicklungsvorgehens der Methodik werden EMF-Modelle für die abstrakte Repräsentation der Modellebenen *SOM-Geschäftsprozessmodell*, *Anwendungsmodell* und *REST-Architekturmodell* sowie für *Modell-Delta*, *Empfehlungsbasis* und *Überarbeitungsschema* verwendet. Das *Implementierungsmodell* wird in Textform (Programmcode) spezifiziert.

Die Definition von *M2M-Transformationen* erfolgt im Eclipse Modeling Project i. d. R. mit der Sprache *Query-View-Transformation* (QVT) [OMG11b] oder der *Atlas Transformation Language* (ATL) [JAB+06]. Beide Ansätze ermöglichen die Realisierung von M2M-Transformationen, wie in der SOM-R-Methodik vorgesehen. Da die QVT als OMG-Standard vorliegt und eine umfassende Dokumentation mit sich bringt, wird sie, bzw. ihre Eclipse-Implementierung, als M2M-Transformationssprache in der vorliegenden Arbeit gewählt (z. B. [OMG11b], [Gron09, S. 549 ff.], [SVE+07, S. 393 ff.]). Der Standard QVT umfasst die drei Komponenten *QVT-Core*, *QVT-Relations* sowie die *Operational Mappings Language* [OMG11b, S. 9 f.]. QVT-Relations unterstützt die Beschreibung von Beziehungen zwischen Modellen in deklarativer Form. Die Operational Mappings Language oder auch *QVT-Operational* ist eine imperative Sprache zur Spezifikation von Mappings zwischen Modellen bzw. ihren Bausteinen. QVT-Core stellt grundlegende Elemente bereit, die von QVT-Operational und -Relations genutzt werden ([OMG11b, S. 9 f.], [SVE+07, S. 204]).

Die dritte und letzte Transformation (T3) der Methodik beschreibt die Ableitung des Programmcodes einer plattformspezifischen Implementierung aus den konsolidierten Schemata des REST-Architekturmodells. Die Spezifikation von M2T-Transformationen wird in Eclipse mit Technologien zur Codegenerierung auf Basis von Templates realisiert. Hierfür stehen z. B. *Java Emitter Templates* (JET), zur Generierung von Java-Klassen, sowie die Templatesprachen *Xpand* und *Xtend* zur Verfügung [Gron09, S. 277 f.]. Für die nachfolgende Realisierung der M2T-Transformationen wird Xpand gewählt, da dieses eine etablierte Technologie ist, mit der Templates zur Codegenerierung in unterschiedlichen Programmiersprachen frei erstellt werden können.

7.1.3 Durchführung von Modellvergleichen

Die Modellvergleichskomponente des Werkzeugprototyps wird auf Basis des Eclipse-Plugins *EMF Compare*[68] realisiert. Mit Compare wird in Eclipse Funktionalität für die Bestimmung von

[68] Offizielle Webseite: http://www.eclipse.org/emf/compare/ (Abruf am 2. März 2015).

Differenzen zwischen zwei oder drei (Versionen von) EMF-Modellen sowie dem Zusammenführen von Modellversionen zur Verfügung gestellt. Im Folgenden wird der Zwei-Wege-Vergleich (Vergleich von zwei Modellen) genutzt (z. B. [Toul06], [BrPi08], [EMFC14]).

Als Ergebnis eines durchgeführten Modellvergleichs erzeugt Compare ein *Differenzmodell*, das die unveränderten sowie die hinzugefügten, entfernten und ähnlichen Bausteine eines „ursprünglichen“ und eines „geänderten“ Modells erfasst. Der generische Vergleichsalgorithmus von Compare ermittelt für die gegenübergestellten Modelle zunächst die übereinstimmenden, gleichen Elemente (matching) und bestimmt auf dieser Basis deren Unterschiede. Als Merkmale werden dabei Name, Inhalt, Typ und Beziehungen der Modellelemente analysiert. Eine Anpassung des Algorithmus an individuelle Anforderungen des vorzunehmenden Modellvergleichs ist möglich. Das erzeugte Differenzmodell ist wiederum selbst ein EMF-Modell und folglich durch die verfügbaren Transformationstechnologien weiterverarbeitbar. EMF Compare erscheint somit geeignet, die Anforderungen des Modellvergleichsansatzes der SOM-R-Methodik zu erfüllen und wird deshalb für die Realisierung der Vergleichskomponente gewählt.

7.2 Komponenten des Entwicklungswerkzeugs

Der Aufbau des Werkzeugprototyps wird anhand der SOM-R-Vorgehensmodelle strukturiert und vorgestellt (Abschnitte 5.4 und 6.6.2). Einen Gesamtüberblick über die einzelnen Komponenten und deren Beziehungen gibt Abbildung 7.2.

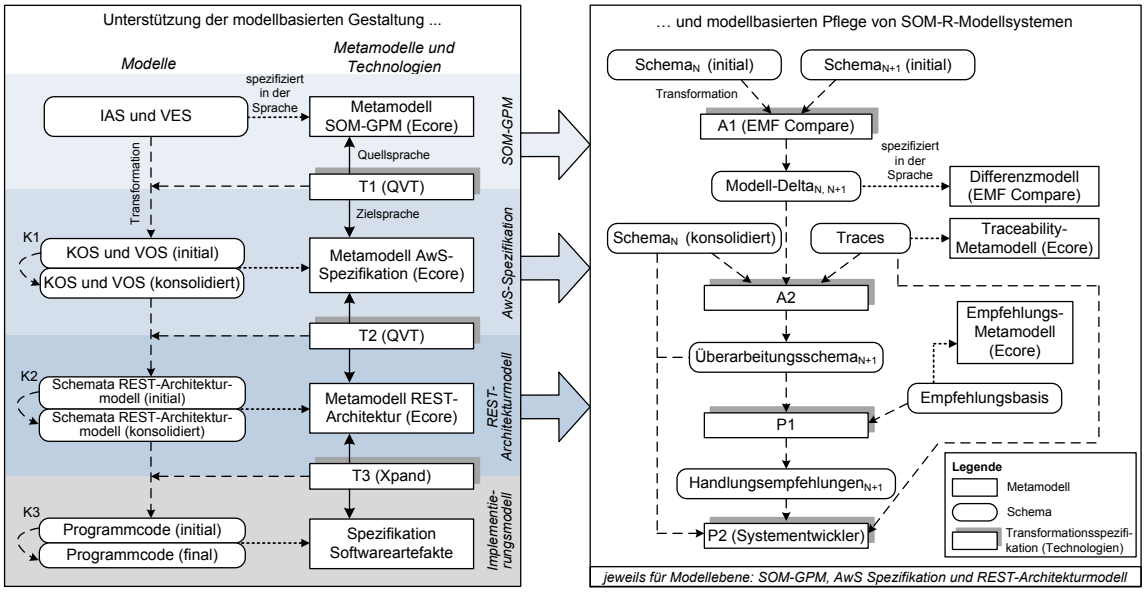


Abbildung 7.2: Komponenten des Werkzeugprototyps

Die modellbasierte Systementwicklung wird bei der Modellbildung (Gestaltung) auf den Ebenen des Modellsystems durch folgende Werkzeugkomponenten (linke Seite) unterstützt:

- Ecore-Metamodelle der Ebenen SOM-Geschäftsprozessmodell, AwS-Spezifikation und REST-Architekturmodell,
- QVT-Transformationsspezifikationen von T1 und T2, sowie
- Xpand-Templates zur Generierung von Programmcode in T3.

Die initialen und konsolidierten Schemata der einzelnen Modellebenen werden jeweils in Form eines EMF-Modells beschrieben. Die Entwicklungsschritte K1, K2 und K3 sind vom Systementwickler unter Einsatz des Werkzeugs (sowie auf Basis des Metamodells) durchzuführen. Eine nähere Erläuterung dieser Schritte erfolgt deshalb nicht (vgl. hierzu Abschnitt 5.5).

Die Implementierung der Modellarchitektur bildet den Ausgangspunkt für die Entwicklung der Komponenten zur Unterstützung der modellbasierten Systempflege. Diese Komponenten realisieren die Funktionalität für die

- Ermittlung von Modelldifferenzen in Analyseschritt A1 auf Basis von EMF Compare, die
- Ecore-Metamodelle zur Spezifikation von Traces und der Empfehlungsbasis, sowie die
- Transformationen zur Ableitung des Überarbeitungsschemas (A2) und der Handlungsempfehlungen (P1).

Die Spezifikation der konsolidierten Schemata aus den Überarbeitungsschemata einer Modellebene (P2) wird wiederum nicht-automatisiert vom Systementwickler durchgeführt. Die Komponenten zur modellbasierten Systempflege werden im Zuge der Durchführung des Weiterentwicklungsprozesses für jede Modellebene durchlaufen (vgl. Abschnitte 6.6.1 und 6.6.2).

7.3 Metamodelle und Transformationen

Die prototypische Implementierung des Entwicklungswerkzeugs sieht die Spezifikation der Metamodelle und Transformationen zur Unterstützung der Entwicklung und Weiterentwicklung von Systemen vor. Die Gestaltung der integrierten Metamodelle auf den Ebenen des SOM-R-Architekturmodells sowie der metamodellbasierten Schematransformation (am Beispiel von T2) erfolgt in Abschnitt 7.3.1. Die Generierung der initialen, plattformspezifischen Implementierung nach dem modellgetriebenen Vorgehen in T3 wird in Abschnitt 7.3.2, und die Metamodelle des Lösungsansatzes zur Systempflege werden in Abschnitt 7.3.3 vorgestellt.

7.3.1 Realisierung der zweiten Transformation (T2)

Die Realisierung der metamodellbasierten Schematransformationen wird am Beispiel der werkzeuggestützten Implementierung des zweiten Transformationsschrittes sowie der Metamodelle der fachlichen AwS-Spezifikation und des REST-Architekturmodells erläutert. Bei der Überführung der konzeptuellen Metamodelle der SOM-R-Methodik in die Ecore-Modelle der Modellierungsplattform ist darauf zu achten, dass der ursprüngliche Aufbau der Metamodelle und die Semantik ihrer Bausteine weitestgehend bewahrt bleiben. Aufgrund plattformspezifischer Rahmenbedingungen ist jedoch stellenweise bei der Gestaltung der Ecore-(Meta)modelle eine Abweichung von der ursprünglich gegebenen Metamodellstruktur erforderlich. Als Beispiele seien die Berücksichtigung des hierarchischen Ecore-Modellaufbaus als Containment-tree (mit Wurzelelement) oder die Überführung des Metaobjekts *Kardinalität* in einen Datentyp genannt (siehe Abbildung 7.3).

Abbildung 7.3 zeigt die Abbildungsbeziehungen von T2 zwischen den Bausteinen des AwS-Metamodells und des REST-Metamodells (Ausschnitt). Aus Gründen einer übersichtlicheren Darstellung werden Metaobjekte zur Beschreibung der Innensicht des REST AwS, wie z. B.

Ausnahmen, *private Methoden* oder *ResponseCode* und *Kardinalität* als Datentypen nicht in der Abbildung erfasst. Nähere Informationen zu den Ecore-Metamodellen beider Ebenen sowie das Metamodell der SOM-Geschäftsprozessebene können beim Autor angefragt werden.

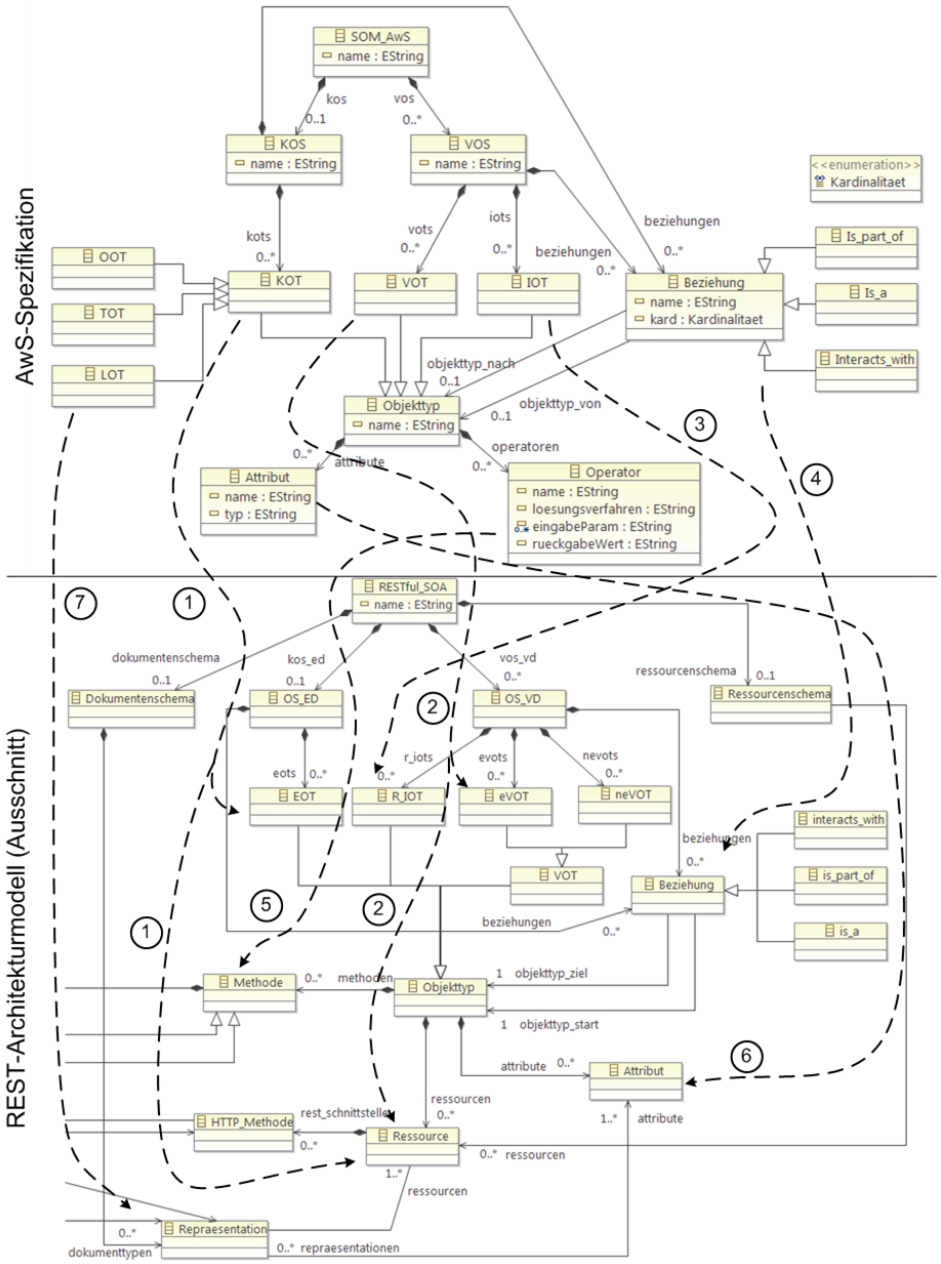


Abbildung 7.3: Definition der metamodellbasierten Schema-Transformation T2 (Ausschnitt)

Ein Modellsystem (EClass SOM_AwS) der Ebene *fachliche AwS-Spezifikation* besteht aus einem KOS sowie beliebig vielen VOS. Jedes Objektschema umfasst wiederum beliebig viele Objekttypen und Beziehungen. Die *Objekttypen* können Attribute und Operatoren besitzen und sind miteinander über Beziehungen verknüpft. Die Kardinalitäten des Metaobjekts *Beziehung* wurden als Aufzählungstyp (enumeration) realisiert. Die Metaobjekte des Metamodells (EClasses) werden in Form von *EAttributes* weitere Eigenschaften zugeordnet (aus Übersichtlichkeitsgründen teilweise ausgeblendet). Meta-Beziehungen zwischen Metaobjekten werden als *EReferences* mit *Containment* (Symbol Kompositionsbeziehung) zur Erstellung des hierarchischen Aufbaus des Modellsystems genutzt.

Auf der Ebene des *REST-Architekturmodells* umfasst das Modellsystem ein OS-ED, beliebig viele OS-VDs sowie ein Dokumenten- und Ressourcenschema. Die Objektschemata beinhalten eine Menge von Objekttypen und die Beziehungen zwischen diesen. EOTs und VOTs (e/ne) können Ressourcen veröffentlichen, die u. a. eine URL (als EAttribut) sowie HTTP-Methoden besitzen und mit beliebig vielen Repräsentationen in Beziehung stehen. Die Repräsentationen können als Request- und Responseparameter öffentliche Methoden zugeordnet werden. Dokumentenschema und Ressourcenschema umfassen eine Menge von Repräsentationen bzw. Ressourcen.

Die metamodellbasierte Schema-Transformation T2 definiert die Abbildung von (1) KOT auf EOT und die Erstellung der zugehörigen Ressourcen, die Abbildung von (2) VOT auf eVOT und zugehörige Ressourcen, (3) IOT auf R-IOT, (4) Beziehungen auf Beziehungen, (5) Operatoren auf Methoden, von (6) Attributen sowie die Überführung von (7) KOTs in Dokumenttypen (Abschnitt 5.5.4). Die Spezifikation der Abbildungsbeziehungen erfolgt im Modellierungswerkzeug mit der Sprache *QVT Operational*. Einen Ausschnitt der erstellten Spezifikationen zeigen die Codefragmente in Quellcode 7.1, Quellcode 7.2 und Quellcode 7.3 für die Überführung von Bausteinen der fachlichen AwS-Spezifikation in das REST-Architekturmodell.

```
// Start der Transformation – Wurzelknoten des Quellmodells wird übergeben.
main() { source.rootObjects()[SOM_AwS]->map Aws2Rest();  }
mapping SOM_AwS :: Aws2Rest() : RESTful_SOA {
  result.name := self.name;
  result.kos_ed := self.kos -> map toKOS_ED()![KOS_ED];
  result.vos_vd += self.vos -> map toVOS_VD();
}
// Mapping von KOS auf OS-ED und Aufruf der Funktionen zur Ableitung
// von EOTs und Beziehungen.
mapping KOS :: toKOS_ED() : KOS_ED {
  result.name := self.name;
  result.eots += self.kots -> map toEOT();
  result.beziehungen += self.beziehungen[Is_a] -> map toGeneralisierung();
  result.beziehungen += self.beziehungen[Is_part_of] -> map toAggregation();
  result.beziehungen += self.beziehungen[Interacts_with] -> map toInteraktion();
}
```

Quellcode 7.1: QVT-Spezifikation von T2 (Beispiel der Erstellung des OS-ED)

Eine QVT-Transformation wird über die *main*()-Funktion gestartet und der Wurzelknoten des Quellmodells (fachliches Anwendungsmodell) der ersten Abbildungsfunktion (*mapping*) übergeben. Jedes Mapping beschreibt eine Transformationsbeziehung zwischen Elementen eines Quell-Metaobjekts und Elementen des Ziel-Metaobjekts. Die Funktion *Aws2Rest*() leitet aus dem Wurzelknoten der fachlichen AwS-Spezifikation (SOM_AwS) den Wurzelknoten der REST-Schemata (RESTful_SOA) ab, übernimmt dessen Namen und ruft Funktionen zur Ableitung von OS-ED und OS-VD aus KOS bzw. VOS auf. Das Mapping *toKOS_ED*() definiert die Zuordnung von KOS und KOS_ED, die im Wesentlichen einen Aufruf der Funktionen zur Transformation von KOT in EOT (*toEOT*()) sowie der verschiedenen Beziehungen zwischen diesen enthält. Eine Überführung des VOS in das OS-VD wird entsprechend der Spezifikation von KOS und OS-ED durchgeführt.

```
// Mapping von KOT auf EOT und Entitätsressourcen
mapping KOT :: toEOT() : EOT {
  result.name := self.name;
  result.ressourcen += createRessources(self.name);
  result.attribute += self.attribute -> map toAttribut();
  result.methoden += self.operatoren -> map toMethode(); }

// Mapping von Attributen und Operatoren des KOT
mapping Operator :: toMethode() : Methode {
  result.name := self.name;
  result.loesungsverfahren := self.loesungsverfahren;
  self.eingabeParam -> forEach(parameter) { result.inputParam += parameter; };
  result.outputParam := self.rueckgabeWert; }
mapping mm_aws::Attribut::toAttribut() : mm_rest_services::Attribut {
  result.name := self.name; result.typ := self.typ; }
```

Quellcode 7.2: QVT-Spezifikation von T2 (Beispiel der Erstellung von EOTs)

Quellcode 7.2 zeigt die Mappings zur Ableitung von EOTs sowie ihren Attributen und Methoden. Die Funktion *toEOT*() transformiert jeden KOT in einen EOT. Den EOTs werden jeweils ein Name zugeordnet und zudem die Funktionen zur Überführung von Attributen und Methoden (*toAttribute*() und *toMethode*()) sowie die Generierung von Ressourcen (*createRessources*()) aufgerufen. Die Mappings der Attribute bzw. Operationen auf Methoden beschreibt die Zuordnung von den Eigenschaften (EAttributes) der Metaobjekte. Nach diesem Schema wird auch die Überführung von IOTs und VOTs des VOS in R-IOTs bzw. eVOTs des OS-VD sowie die Erzeugung von Repräsentationen auf Basis der Attribute von KOTs durchgeführt.

```
// Helper zur Generierung von Entitätsressourcen
helper createRessources(bez : String) : Sequence(Ressource) {
  return Sequence {
    object Ressource {
      name          := 'ER_' + bez;
      ressourcenURL   := '/' + bez.toLowerCase() + '/';
      ressourcenTyp   := 'Listenressource';
      rest_schnittstelle += createLRSchnittstelle(ressourcenTyp);      },
    object Ressource {
      name          := 'ER_' + bez;
      ressourcenURL   := '/' + bez.toLowerCase() + '/{id}/';
      ressourcenTyp   := 'Individualressource';
      rest_schnittstelle += createIRSchnittstelle(ressourcenTyp);
    } };
}
```

Quellcode 7.3: QVT-Spezifikation von T2 (Beispiel der Erstellung von Ressourcen)

Die Hilfsfunktion *createRessources*() wird im Rahmen der Ableitung von EOTs aufgerufen und spezifiziert die Erzeugung von Individual- (IR) und Listenressourcen (LR) als Objektmenge (Sequence) (siehe *toEOT*() in Quellcode 7.2). Als Ressourcen-Eigenschaften werden Name, URL und Typ zugeordnet und anschließend die Hilfsfunktionen *createLRSchnittstelle*() und *createIRSchnittstelle*() mit dem Parameter des Ressourcentyps aufgerufen. Diese generiert nach demselben Prinzip die HTTP-Methoden für die Individual- und Listenressourcen.

Der vorgestellte Auszug des SOM-R-Architekturmodells und der Schema-Transformationen vermittelt einen Eindruck der prototypischen Werkzeugrealisierung in EMF. Auf Basis der gewählten Technologien sind die automatisierten Entwicklungsschritte des konzipierten Lösungsansatzes zur Systementwicklung somit durchgängig umsetzbar.

Die Modellbildung mit dem Werkzeugprototyp wird am Beispiel der AwS-Spezifikation der Fallstudie verdeutlicht. Abbildung 7.4 zeigt einen Ausschnitt der konsolidierten Flugbuchungsschemata.

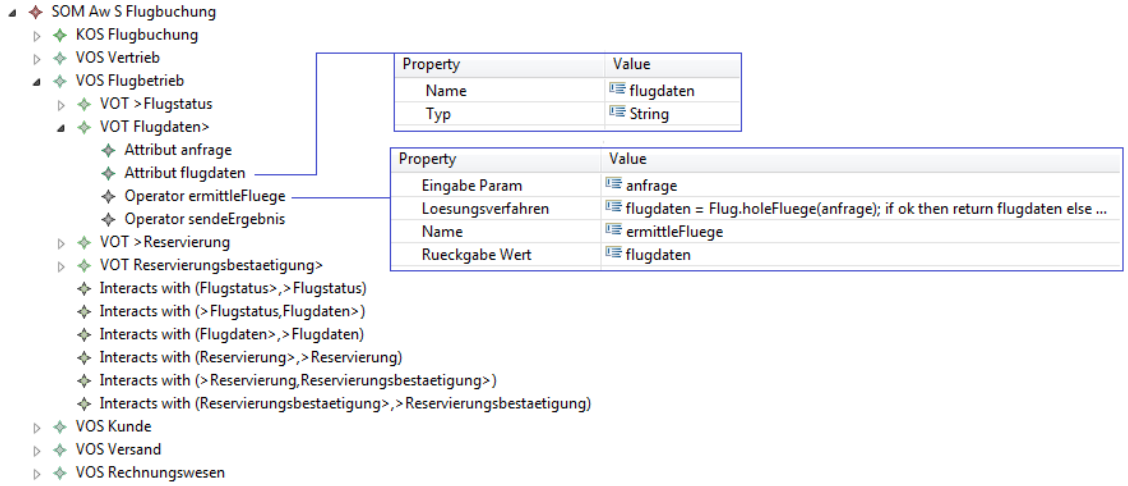


Abbildung 7.4: Einsatz des Modellierungswerkzeugs am Beispiel der Fallstudie

Der VOT *Flugdaten>* des VOS *Flugbetrieb* realisiert seine Funktionalität durch die internen Attribute *Anfrage* und *Flugdaten* sowie die Operatoren *ermittleFluege* und *sendeErgebnis* (Abbildung 5.13). Der Zugriff auf einen Ausschnitt des KOS ist in den Lösungsverfahren der Operatoren beschrieben. Zwischen VOTs bestehen interacts_with-Beziehungen, die ebenfalls Bausteine des VOS sind. Weitere Informationen zu den Modellen der Fallstudie, den Skripten der Transformationen T1 und T2 sowie den Ecore-Metamodellen der Modellebenen können beim Autor dieser Arbeit angefragt werden.

7.3.2 Generierung der plattformspezifischen Implementierung

Die Generierung von Programmcode der plattformspezifischen Implementierung ist das Ziel der dritten (und letzten) Transformation. Der Schritt T3 markiert somit den Wechsel von einer grafischen, modellbasierten in eine textuelle Repräsentationsform. Hierbei erfolgt keine explizite Erstellung des Implementierungsmetamodells. Die Realisierung der M2T-Transformation im Werkzeugprototyp nutzt Templates, die in der Sprache *Xpand* definiert sind (Abschnitt 7.1.2). Nach diesem Ansatz werden die Elemente des (EMF-)Quellmodells (REST-Architekturmodell) in „Platzhalter“ der Template-Vorlage eingebettet, im Generierungslauf instanziiert (eingesetzt) und dadurch Programmcode erzeugt.

Der Einsatz von Templates wird am Beispiel der Erzeugung von Java-Klassen zur Realisierung von Entitätsdiensten erläutert (Abschnitte 5.5.6 und 5.5.7). Quellcode 7.4 zeigt drei Xpand-Templates, welche die Generierung des Java-Interface (javaInterfaceEOT) und der Implementierungsklasse (javaClassImplEOT) sowie der Entitätsklasse (javaEntity) für die EOTs des OS-ED beschreiben. Beginn und Ende einer Templatedefinition markieren die Schlüsselworte *DEFINE* und *ENDDEFINE*. Templatespezifische Ausdrücke sowie die referenzierten Elemente eines EMF-Modells werden von Guillemets (« ») umschlossen.

Im dargestellten Beispiel erzeugt das Template *javaInterfaceEOT* für jeden EOT eine Java-Datei auf Basis der definierten Vorlage. Dabei werden die referenzierten Modellbausteine, wie z. B. die Benennung des Interface mit EOT-Name («name») gefüllt. Anschließend wird für jede Ressource des EOT ein weiteres Template zur Spezifikation der REST-Schnittstelle aufgerufen (*ressource_tmplt*). Jedes Java-Interface wird von jeweils einer Implementierungsklasse implementiert, deren (vereinfachten) Aufbau das Template *javaClassImplEOT* zeigt. So wird beispielsweise für jede Methode (Metaobjekt *Methode*) eine neue Java-Methode angelegt und ihre Signatur spezifiziert. Das Template *javaEntity* beschreibt die Generierung einer Entitätsklasse mit den Attributen des EOT und zugehörigen Getter- und Setter-Methoden.

```
«DEFINE javaInterfaceEOT FOR EOT»
    «FILE name+"IF.java"»
    import javax.ws.rs.*; // weitere imports …
    @Path("/«name»/")
    public interface «name»IF {
        «EXPAND ressource_tmplt FOREACH ressourcen»
    } «ENDFILE» «ENDDEFINE»

«DEFINE javaClassImplEOT FOR EOT»
    «FILE name+"Impl.java"»
    import de.flugbuchung.entitaet.«name»; // weitere imports …
    public class «name»Impl implements «name»IF {
       «FOREACH methoden AS m»
       private «m.outputParam.typ» «m.outputParam» «m.name» («m.inputParam»){
         «m.loesungsverfahren»; //Lösungsverfahren einfügen
         return «m.outputParam»;
       } «ENDFOREACH»
    } «ENDFILE» «ENDDEFINE»

«DEFINE javaEntity FOR EOT»
  «FILE name+".java"»
    import javax.xml.bind.annotation.XmlRootElement;
    @XmlRootElement
    public class «name» {
        «FOREACH attribute AS a»
        private «a.typ» «a.name»;
        «ENDFOREACH»
        //getter, setter
        «FOREACH attribute AS a»
        public «a.typ» get«a.name»() { return «a.name»; }
        public void set«a.name»(«a.typ» «a.name») { this.«a.name» = «a.name»; }
        «ENDFOREACH»
    } «ENDFILE» «ENDDEFINE»
```

Quellcode 7.4: Ausschnitt des Xpand-Templates von T3

Entsprechend dem Vorgehen der Erzeugung von Programmcodes für die Entitätsdienste können auch die Klassen der Vorgangsdienste sowie die Konfigurationsklassen von Applikations- und Datenbankserver generiert werden. Einen Eindruck der erzeugten Java-Klassen gibt das Implementierungsmodell (Abschnitt 5.5.8.2).

7.3.3 Modellbildung im Rahmen der Systemweiterentwicklung

Die Durchführung der Weiterentwicklungsaufgabe in der SOM-R-Methodik erfordert die modellbasierte Spezifikation der existenziellen Abhängigkeitsbeziehungen (Traces) zwischen Modellobjekten, den Überarbeitungsschemata der Modellebenen sowie der Empfehlungsbasis (Abbildung 7.2). Die Metamodelle für die genannten Bestandteile des Lösungsansatzes der

Systempflege werden nachfolgend erläutert. Das Erstellen der Metamodelle erfolgt ebenfalls in Ecore, um durch die Nutzung einer gemeinsamen Plattform eine integrierte Verarbeitung der Modelle zu unterstützen.

Dokumentation von Entwicklungsinformationen

Der Dokumentationsansatz definiert das Erfassen der Schemata auf den Modellebenen des Architekturmodells sowie der strukturellen Abhängigkeitsbeziehungen, die im Rahmen der Modellbildung entstehen. Die strukturellen Abhängigkeitsbeziehungen zwischen den Bausteinen des SOM-R-Modellsystems werden auf Basis eines Traceability-Metamodells dokumentiert. Abbildung 7.5 zeigt das Ecore-Metamodell, welches entsprechend den Vorgaben des Dokumentationsansatzes (Abschnitt 6.3.2.2) gestaltet wurde.

Ein Eintrag im Trace-Modell umfasst ein oder mehrere Modellobjekte und maximal eine Abhängigkeitsbeziehung. Der Aktivitätszeitpunkt von Einträgen ab einer bestimmten Schema-Version sowie seine optionale Inaktivität werden in Attributform erfasst. Jedes Modellobjekt referenziert ein zu diesem korrespondierendes Objekt in den Schemata des Modellsystems und besitzt einen eindeutigen Identifikator. Abhängigkeitsbeziehungen sind vom Typ „Transformation“ oder „Konsolidierung“ und beschreiben Vor-/Nachfolgerbeziehungen zwischen zwei Objektmengen.

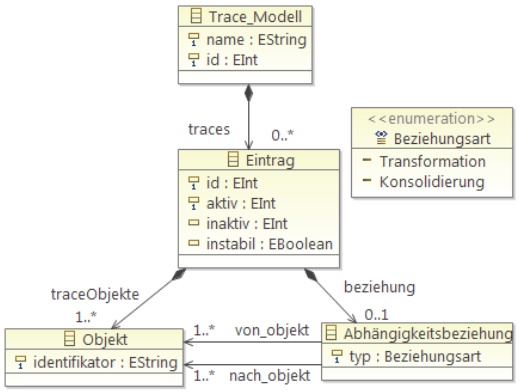


Abbildung 7.5: Traceability-Metamodell

Die Einträge des Trace-Modells, welche eine Transformation zwischen Modellobjekten dokumentieren, sind im Zuge der Durchführung der QVT-Mappings automatisiert erzeugbar. Die Spezifikation von Traces der Konsolidierungsschritte ist z. B. auf Basis von „Listeners“ im Entwicklungswerkzeug realisierbar, um die manuellen Änderungen an einem Schema in Form von Konsolidierungsbeziehungen automatisiert zu erfassen.

Sowohl die entwickelten initialen und konsolidierten Schemata, als auch die Abhängigkeitsbeziehungen liegen in Form von EMF-Modellen vor. Für die Persistierung, Versionierung und Verwaltung der spezifizierten Modelle ist ein Modell-Repositorium heranzuziehen. Im Umfeld des Eclipse Modeling Project unterstützen aktuell das *CDO Model Repository*[69] oder *EMFStore*[70]

[69] Offizielle Webseite von CDO (Connected Data Objects): http://www.eclipse.org/cdo/ (Abruf am 2. März 2015).

[70] Offizielle Webseite von EMFStore: http://eclipse.org/emfstore/ (Abruf am 11. Januar 2015).

eine Durchführung dieser Aufgabe und können für eine Realisierung herangezogen werden. Nähere Informationen zu den genannten Technologien finden sich auf ihren Projektwebseiten.

SPEZIFIKATION DES MODELL-DELTAS

Gegenstand des ersten Analyseschrittes (A1) ist die Durchführung von Modellvergleichen. Sie werden im Werkzeugprototyp mit EMF Compare realisiert. Die Unterschiede und Gemeinsamkeiten zwischen zwei Modellen spezifiziert ein Differenzmodell. Eine stark vereinfachte Darstellung des Compare-Metamodells, welches diesem zugrunde liegt, zeigt Abbildung 7.6.

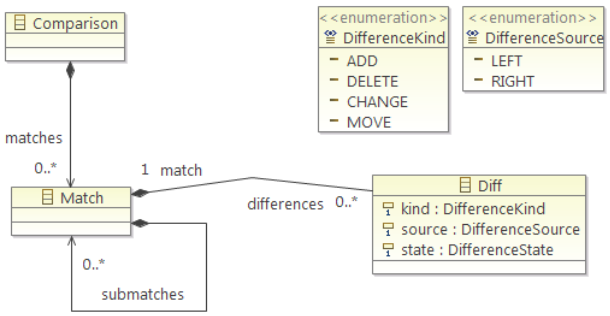


Abbildung 7.6: Ausschnitt des Compare-Metamodells[71]

Das Ergebnis eines Modellvergleichs besteht aus einer Menge von *Matches*, die für jeden Modellbaustein der verglichenen Modelle die *Übereinstimmung* zweier Elemente repräsentieren. Matches besitzen eine Referenz auf die Objekte in Originalmodell sowie dem geänderten Modell, oder markieren einen *Unterschied* und referenzieren damit nur ein Objekt. Das *Originalmodell* wird als linkes und das *geänderte Modell* als rechtes Modell bezeichnet. Die Ermittlung von Matches stellt den ersten Vergleichsschritt dar. Dabei können auch für ähnliche oder verschobene Bausteine Übereinstimmungen ermittelt werden, was in der vorliegenden Arbeit jedoch nicht weiter betrachtet wird. Auf Grundlage des erstellten „matching model“ werden anschließend die Unterschiede zwischen (ähnlichen) Matches berechnet und die Änderungen am Originalmodell bestimmt. Ergebnis dieses Schrittes ist das „difference model“. Die Ermittlung des Modell-Deltas zwischen zwei Schemaversionen ist somit auf Basis von EMF Compare unter Einhaltung der Vorgaben des SOM-R-Modellvergleichsansatzes realisierbar. Da das hierbei erzeugte Differenzmodell als EMF-Modell vorliegt, kann seine Weiterverarbeitung unter Einsatz der bereits eingeführten Transformationstechnologien erfolgen.

Die Ermittlung des Modell-Deltas wird am Beispiel des Entfernens der betrieblichen Transaktion *Lieferbestätigung* verdeutlicht (Abbildung 6.29). Der Modellvergleich wird auf der Ebene der fachlichen AwS-Spezifikation anhand des „aktualisierten“ initialen KOS und der initialen VOS (Version N+1) mit den ursprünglichen Schemata (Version N) verglichen. Das Ergebnis des Modellvergleichs zeigt Abbildung 7.7.

[71] Match- und Diff-Metamodell von EMF Compare sind Teil des Eclipse-Plugins *org.eclipse.emf.compare*.

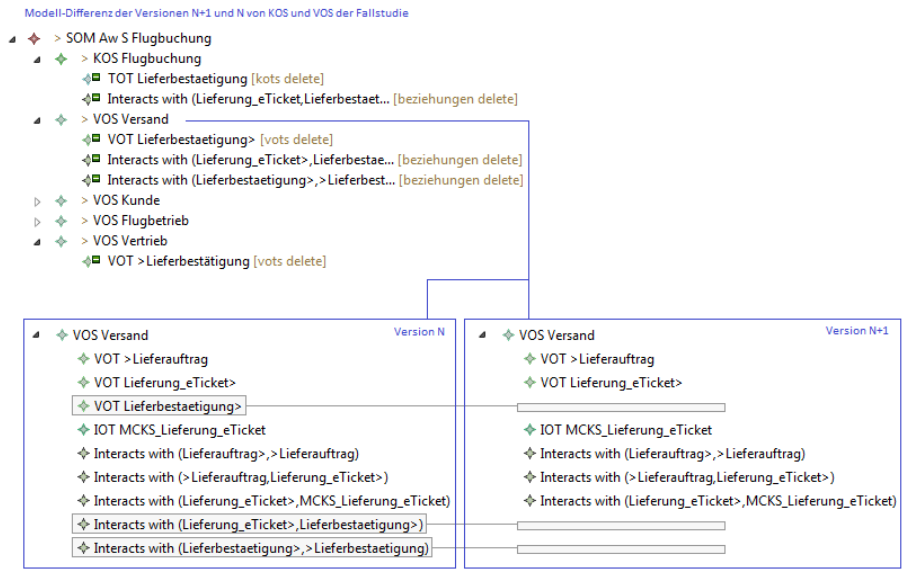


Abbildung 7.7: Ermittelte Modell-Differenz am Beispiel der entfernten Transaktion *Lieferbestätigung* der Fallstudie (Vergleich von Version N+1 und N)

Die Unterschiede der Schemata werden in Form von Differenzeinträgen beschrieben. Im Beispiel wurden der Objekttyp *TOT:Lieferbestätigung* und die interacts_with-Beziehung zu *TOT:Lieferung_eTicket* aus dem KOS gelöscht. Korrespondierend dazu wurden auch die Containment-Beziehungen ([kots], [beziehungen]) zwischen TOTs bzw. interacts_with-Beziehung und KOS aus dem Modell entfernt. Die Differenzeinträge spezifizieren somit die Transformationsschritte zur Überführung der Modelle von Version N in Version N+1. Wie bereits ausgeführt, ist das erzeugte Modell-Delta ein EMF-Modell, das Referenzen auf die Bausteine der verglichenen Modelle enthält. Auf dieser Basis kann das Erstellen der Überarbeitungsschemata erfolgen.

METAMODELL DER ÜBERARBEITUNGSSCHEMATA

Die Überarbeitungsschemata einer Modellebene stellen das Ergebnis des zweiten Analyseschrittes (A2) dar. Das konsolidierte Schema der ursprünglichen Modellversion N wird dabei um die Änderungen des angepassten initialen Schemas angereichert. Die instabilen Modellbausteine des Überarbeitungsschemas werden im Zuge der Analyse mit einer Markierung des Typs ihrer Veränderung bzw. Instabilität (Differenzart) versehen.

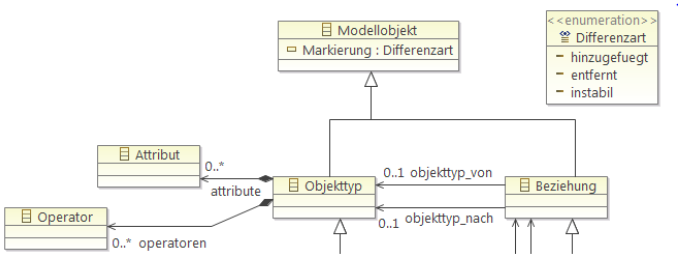


Abbildung 7.8: Metamodell des Überarbeitungsschemas der fachlichen AwS-Spezifikation (Ausschnitt)

Das Metamodell der Überarbeitungsschemata der fachlichen AwS-Spezifikation zeigt Abbildung 7.8. Es spezifiziert eine Erweiterung des Metamodells der AwS-Modellebene um das Metaobjekt *Modellobjekt* mit dem Attribut *Markierung*, welches Supertyp der Modellbau-

steine des konzeptuellen Metamodells ist (vgl. Metaobjekte in Abschnitt 6.3.2.1). Ein Modellobjekt kann als *hinzugefügt*, *entfernt* oder *instabil* markiert werden. Im Falle von KOS und VOS sind dies die Objekttypen (KOT, VOT und IOT) sowie die Beziehungen zwischen diesen.

Das Metamodell des Überarbeitungsschemas auf der Ebene des REST-Architekturmodells stellt Abbildung 7.9 dar. Die markierten Modellobjekte sind Objekttypen (EOT, eVOT, neVOT, R-IOT, DOT), die Beziehungen zwischen den Objekttypen sowie Ressourcen und Repräsentationen.

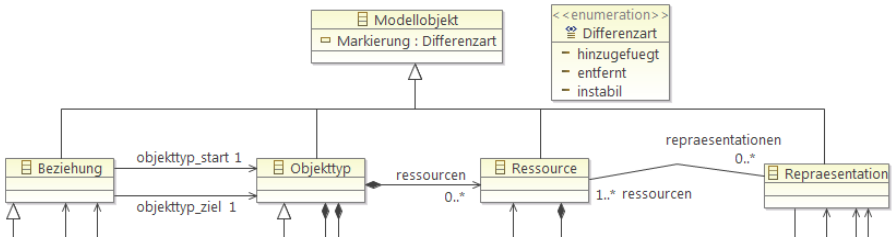


Abbildung 7.9: Metamodell des Überarbeitungsschemas des REST-Architekturmodells (Ausschnitt)

Grundlage für die Durchführung der Modellbildung in Schritt A2 sind die Metamodelle zur Spezifikation der Überarbeitungsschemata. Das Überarbeitungsschema selbst ist ein EMF-Modell. Auf Basis der eingesetzten Transformationstechnologien ist auch die automatisierte Erzeugung der Schemata aus den EMF-Modellen der Modell-Differenz, der konsolidierten $Schemata_N$, der initialen $Schemata_{N+1}$ sowie den dokumentierten Abhängigkeitsbeziehungen realisierbar.

Die beispielhafte Modellierung der Überarbeitungsschemata zeigt Abbildung 7.10 für den VOS *Versand* der Fallstudie (Abbildung 6.30). Der VOT *Lieferbestätigung*> sowie die Interacts_with-Beziehung von VOT *Lieferung_eTicket*> sind als „entfernt“ markiert. Eine „Instabilitätsmarkierung“ besitzt der VOT *Lieferung_eTicket*>, da dieser mit einem geänderten Modellbaustein (entfernte Interact_with-Beziehung) verknüpft ist.

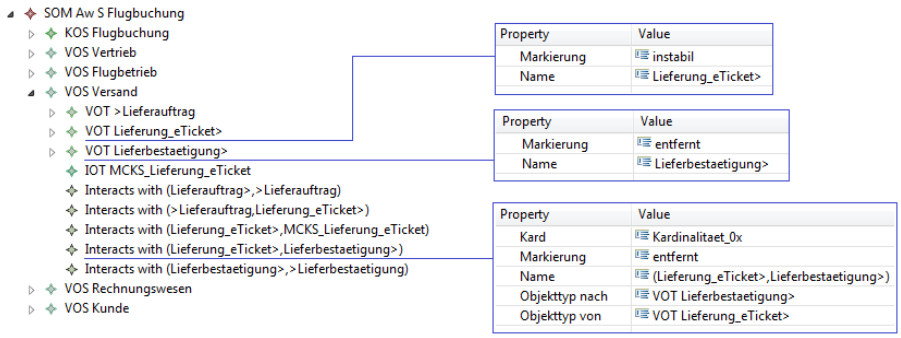


Abbildung 7.10: Überarbeitungsschema des VOS *Versand* der Fallstudie (Beispiel)

SPEZIFIKATION DER EMPFEHLUNGSBASIS

Das Ziel des ersten Pflegeschrittes (P1) ist die Ableitung von Empfehlungen für die Konsolidierung der instabilen Modellobjekte in den Überarbeitungsschemata. Im Entwicklungswerkzeug wird die Spezifikation der Empfehlungsbasis ebenfalls in Modellform auf Basis von EMF umgesetzt. Die konzeptuellen Anforderungen an die Implementierung beschreibt der SOM-R-Empfehlungsansatz (Abschnitt 6.5.2). Das Metamodell des Empfehlungsansatzes zeigt Abbildung 7.11.

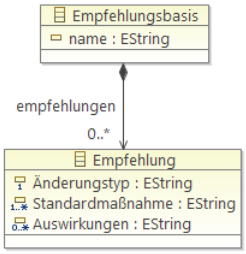


Abbildung 7.11: Metamodell der Empfehlungsbasis

Eine Empfehlungsbasis besteht aus einer Menge von Empfehlungen, die für einen Änderungstypen die Standardmaßnahmen und Auswirkungen erfassen. Das Attribut *Änderungstyp* dient als Identifikator zur Selektion von Handlungsempfehlungen für ein bestimmtes Modellobjekt mit Differenz-/Änderungsmarkierung (z. B. + KOT, siehe Abschnitt 6.5.2). Die Spezifikation von Standardmaßnahmen und Auswirkungen erfolgt ebenfalls in Form eines Attributes des Metaobjekts *Empfehlung*. Die ermittelten Empfehlungen können somit dem Entwickler in Textform zur Verfügung gestellt werden.

Die Modellierung der Empfehlungsbasis zeigt Abbildung 7.12 am Beispiel der initialen Version von Empfehlungen zur Änderungsbehandlung auf der Ebene der AwS-Spezifikation. Anhand der Eigenschaft *Änderungstyp* kann eine Ermittlung der Empfehlungseinträge passend zur Änderung im Überarbeitungsschema erfolgen. Ein Zugriff auf die Empfehlungsbasis ist ebenfalls unter Einsatz der Transformationssprache realisierbar.

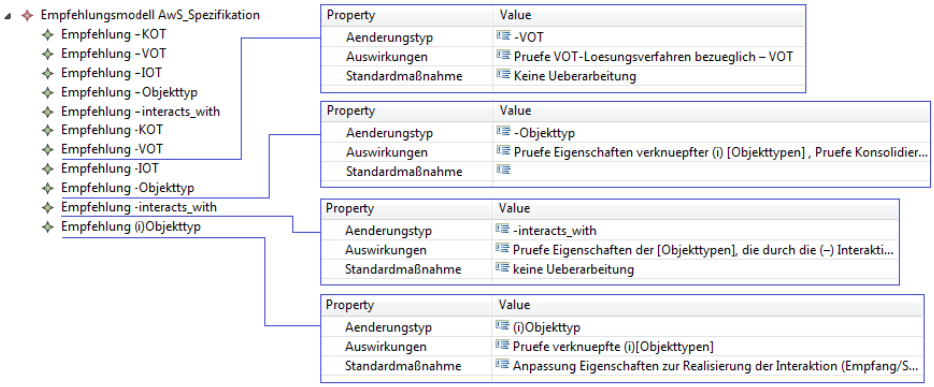


Abbildung 7.12: Initiale Empfehlungsbasis der AwS-Spezifikation (Beispiel)

Die Ableitung von Handlungsempfehlungen stellt den letzten automatisiert durchführbaren Schritt im Lösungsansatz der Systempflege dar. Die pflegende Schema-Konsolidierung (Schritt P2) wird manuell vom Entwickler am Überarbeitungsschema vorgenommen. Das Ergebnis dieses Schrittes sind die neuen Versionen (N+1) der konsolidierten Schemata einer Modellebene. Die Anpassung des Überarbeitungsschemas erfolgt ebenfalls im Werkzeug. Der Systementwickler wird bei der Durchführung der Weiterentwicklungstätigkeit zudem durch Assistenzinformationen in Form grafischer oder textueller Repräsentationen der modellierten Abhängigkeitsbeziehungen, Empfehlungen sowie den konsolidierten und den initialen Schemata unterstützt.

7.4 Zusammenfassung

Die modellbasierte Spezifikation der Ebenen des SOM-R-Architekturmodells ist eine komplexe Aufgabe, deren Durchführung durch den Einsatz eines Entwicklungswerkzeugs unterstützt werden kann. Mit der Implementierung des Softwareprototyps wird in diesem Kapitel der Machbarkeitsnachweis einer werkzeuggestützten Anwendung der SOM-R-Methodik erbracht. Dieser demonstriert die Umsetzung der konzipierten Metamodelle sowie der ausgewählten Transformationsspezifikationen der Methodik. Die Realisierung der einzelnen Lösungsansätze zur modellbasierten Gestaltung und Pflege von Systemen erfolgt auf Basis der Metamodellierungsplattform EMF.

Die Unterstützung der SOM-R-Methodik durch das prototypische Modellierungswerkzeug wird für die Aufgabe der „Neu“-Entwicklung von RESTful SOA aus SOM-Geschäftsprozessmodellen demonstriert. Mit den Metamodellen zur Spezifikation von Traces, Empfehlungsbasis und Überarbeitungsschemata werden zudem die Bestandteile für die Modellbildung zur Durchführung der Systemweiterentwicklungsaufgabe erarbeitet. Die Ausführungen werden am Beispiel der Schemata KOS und VOS aus der Fallstudie verdeutlicht. Die Realisierung der Funktionalität zur Verwaltung von Modellversionen, der automatisierten Erfassung von Abhängigkeitsbeziehungen im Rahmen der Schema-Konsolidierung, der Umsetzung des Algorithmus zur Ermittlung des Überarbeitungsschemas sowie der Ableitung von Empfehlungen werden dabei beleuchtet. Um die Implementierbarkeit auch für diese Funktionsbereiche darzulegen, werden geeignete Lösungsansätze vorgestellt, auf deren Basis eine Werkzeugunterstützung umsetzbar ist.

8 Verwandte Arbeiten

Im achten Kapitel werden verwandte Ansätze aus dem Themenbereich der Arbeit untersucht. Zunächst wird der Stand der Literatur für den Betrachtungsgegenstand RESTful SOA und ihre modellbasierte Spezifikation aus Geschäftsprozessmodellen beleuchtet. Anschließend erfolgt ein Vergleich der SOM-R-Methodik mit ausgewählten Ansätzen zur Entwicklung von SOA auf Basis der Model Driven Architecture sowie einer umfassenden SOA-Entwicklungsmethode. Die Ausführungen schließen mit der Erläuterung von Konzepten für eine modellgetriebene Anpassung von Softwaresystemen. Parallel zu der inhaltlichen Betrachtung des Themenbereichs wird der Mehrwert des Lösungsansatzes der vorliegenden Arbeit herausgearbeitet.

8.1 Modellbasierte Spezifikation von Systemen und der Architekturstil REST

Die tiefergehende Betrachtung verwandter Ansätze zu den einzelnen Bestandteilen des vorliegenden Untersuchungsobjekts beginnt mit der modellbasierten Spezifikation von Systemen im Kontext von REST. Um einen Überblick über den aktuellen Stand der Literatur zur *modellbasierten Entwicklung (Spezifikation)* von *REST-Anwendungen* auf Basis von *Geschäftsprozessmodellen* zu erhalten, wird dieser Gegenstand in drei Themenbereiche gegliedert:

- Ansätze zur modellgetriebenen/-basierten Spezifikation von REST-Anwendungen
- Modellierungsansätze für REST-Anwendungen oder die REST-Schnittstelle
- Ansätze zur Modellierung von RESTful „Business Processes“

Die für eine Untersuchung ausgewählten modellgetriebenen REST-Entwicklungsansätze werden dem konzipierten Vorgehen zur Modellbildung in der SOM-R-Methodik sowie dem Modellierungsansatz des REST-Architekturmodells gegenübergestellt (Kapitel 5). Die Betrachtung der REST-spezifischen Arbeiten schließt mit einer Einordnung von Beiträgen zur RESTful Modellierung von Geschäftsprozessen.

8.1.1 Ansätze zur modellbasierten Entwicklung von REST-Anwendungen

Mit der SOM-R-Methodik wird ein Ansatz zur *modellbasierten Entwicklung von RESTful SOA* vorgeschlagen. Tabelle 8.1 listet Arbeiten auf, die dieses Ziel ebenfalls verfolgen. Die verwandten Arbeiten werden anhand folgender vier Merkmale genauer beschrieben:

- Eingesetzte Modellierungssprachen für die Spezifikation von Quell- und Zielmodellen
- Spezifizierte Abstraktionsebenen der Quell- und Zielmodelle (fachliche (f), softwaretechnische (sw) und Implementierungs-Ebene (i))
- Erreichter Automatisierungsgrad des Entwicklungsprozesses (automatisierte oder teilautomatisierte Überführung der Quell- in Zielmodelle)

- Erfasste Systemsichten (Daten, Interaktion, Funktion, Ablauf) [FeSi13, S. 142 f.]

Zudem wird das in den Ansätzen dargelegte Entwicklungsvorgehen in knapper Form erläutert.

Autor	Quellmodell	Zielmodell	Beschreibung
[LSS09], [LKS06], [SLS+08]	Modell funktionaler Service-Anforderungen (UML-Sequenzdiagramm). *Zwischenmodell*: Information Model der identifiz. Entitäten (UML-Klassendiagramm). *Modellebene*: f, sw	Ressourcenmodell (UML-Klassendiagramm) und WADL-Beschreibung der RESTful WS. *Modellebene*: sw, i	Sequenzdiagramme beschreiben funktionale Service-Anforderungen in Form relevanter Interaktionen zw. Client u. Server (Analysephase). Auf Basis dieses Modells werden Ressourcen identifiziert (information model), anschließend ein Ressourcenmodell abgeleitet u. daraus ein WADL-Dokument generiert. *Vorgehen*: teilautomatisiert, modellbasiert *Fokus*: Daten, Interaktion, Funktion
[RRS+10], [PoRa11]	Struktur u. Verhalten der RESTful WS werden mittels Conceptual Models (Klassen-, Sequenzdiagramm), Process Model (UML-Aktivitätsdiagramm) und Behavioral Model (UML-Zustandsdiagramm) erfasst. *Modellebene*: sw	Nicht näher spezifiziert. Die Transformation in eine Implementierung mittels BPEL4REST, WADL und Ruby on Rails wird angegeben, jedoch nicht weiter vorgestellt. *Modellebene*: i	Modellierung der Spezifikation von RESTful WS und ihrer Komposition auf sw-techn. Ebene. Konzentration auf den Einsatz von UML zur Beschreibung statischer u. dynamischer Merkmale von RESTful WS. *Vorgehen*: automatisiert, 'Model-Driven' (nicht näher spezifiziert) *Fokus*: zentrale Struktur und Verhaltenseigenschaften
[PDM+11]	Domain Model u. Navigational Model (Ecore) zur Beschreibung der Entitäten u. Operationen von RESTful WS (PIM). *Modellebene*: sw	Spezifikation der REST-Schnittstelle in einem Application Facade Component Model (PSM). *Modellebene*: i	Transformation von Domain u. Navigational Model des RESTful WS in das Component Model ist mit QVT spezifiziert. Die DSLs der Quellmodelle basieren auf dem OOH4RIA Metamodell [MGP+08]. *Vorgehen*: automatisiert, MDA *Fokus*: Daten, Interaktion
[OLA+12]	UML-Profil zur Spezifikation von RESTful WS (erweit. UML-Klassendiagramm). *Modellebene*: sw, i	Java-Klassen (Controller) und Roo Script (Entitäten) der RESTful WS. *Modellebene*: i	Implementierungsnahe Modellierung von Controllern, Views u. Entitäten der RESTful WS auf Basis eines UML Profils. Aus den erweiterten Klassendiagrammen werden die Java-Klassen des Controllers und Roo Scripts der Entitäten generiert. *Vorgehen*: automatisiert, MDA *Fokus*: Daten, Interaktion
[TaVa13]	WSSR-Modelle (Klassendiagramme) als PIM der REST-Schnittstelle. Vorstellung eines eigenen WSSR-Metamodells (RESTful Semantic Web Service). *Modellebene*: sw	Formale Spezifikation der REST-Schnittstelle (PSM). Als Zielsprachen werden WADL, WSDL 2.0, OWL-S und SA-REST genannt. *Modellebene*: i	Modellgetriebene Ableitung formaler Schnittstellenbeschreibungen von RESTful WS aus dem WSSR-Modell. Die Verknüpfung zw. Services u. Ontologien ist modellierbar. *Vorgehen*: automatisiert, MDA. *Fokus*: Schnittstelle (Daten)

Tabelle 8.1: Ansätze zur modellbasierten Entwicklung von REST-Anwendungen

Grundsätzlich lässt sich in den vorgestellten Entwicklungsansätzen eine Fokussierung der modellbasierten Beschreibung von RESTful Web Services (RESTful WS) auf die softwaretechnische Ebene feststellen (z. B. [PDM+11], [TaVa13], [RRS+10], [PoRa11]). Dabei wird meist von dem eigentlichen Erstellungsprozess des Modells abstrahiert und lediglich die Transformation des Quellmodells in eine implementierungsnahe Spezifikation untersucht. Das MDA Framework dominiert dabei als Architektur der modellgetriebenen Entwicklung (z. B. [TaVa13], [OLA+12], [PDM+11]).

Einen Vorschlag zur detaillierten Struktur- und Verhaltensmodellierung von RESTful Web Services mit UML-Diagrammen unterbreiten PORRES ET AL ([RRS+10], [PoRa11]). Die Autoren betrachten jedoch nicht die Transformation aus oder in andere Abstraktionsebenen.

Ein expliziter Bezug auf fachliche Anforderungen als Quelle von RESTful WS findet sich in den untersuchten verwandten Arbeiten lediglich bei LAITKORPI ET AL ([LSS09], [LKS06]). Die Autoren nennen Service-Anforderungen als Startpunkt des Entwicklungsprozesses und beschreiben ihre Überführung in eine funktionale Spezifikation des Zielsystems (Sequenzdiagramm) als Aufgabe der Analysephase. Die modellierte Funktionalität des Systems wird anschließend auf Klassen und ihre elementaren Operationen heruntergebrochen (Phase behavioral canonicalization). Das Ergebnis ist ein „Information Model", welches als Basis für die Identifikation von Ressourcen mit einheitlicher Schnittstelle im „Resource Model" dient (Phase structural canonicalization). Das Resource Model kann wiederum in eine formale Schnittstellenspezifikation transformiert werden. Der vorgeschlagene Ansatz ist modellbasiert. Er definiert den Einsatz von Modelltransformationen jedoch erst auf der Softwareebene [LSS09, S. 175 ff.].

Die vorgestellten modellbasierten Entwicklungsansätze beschäftigen sich nur am Rande mit dem Prozess der *Identifikation von Ressourcen auf hohen Abstraktionsebenen*. Eine verbreitete Vorgehensbeschreibung zur Identifikation und Entwicklung von RESTful Services stammt von RICHARDSON UND RUBY [RiRu07]. Die Aufdeckung von Ressourcen erfolgt hierbei ausgehend von Anwendungsfällen und (natürlichsprachlich formulierten) fachlichen Anforderungen. Auf Basis von Nomen findet das Aufdecken von Entitäten des Systems statt. Diese werden anschließend in Ressourcen überführt [RiRu07, S. 107 ff.]. Auch wenn diesem Vorgehen kein modellbasierter Ansatz zugrunde liegt, so stellt es einen wichtigen Beitrag zur Ressourcenidentifikation dar, an den sich eine Vielzahl von REST-Arbeiten anlehnt (z. B. [Burk13], [Tilk11], [Alla10], [LSS09]).

Zusammenfassend ist festzuhalten, dass die untersuchten Ansätze zur modellbasierten Spezifikation/Entwicklung von RESTful SOA die Herausforderung einer Überwindung der semantischen Lücke zwischen fachlichen Modellen auf hohen Abstraktionsebenen und der softwaretechnischen Spezifikation methodisch nur defizitär lösen. Dies schließt auch die Aufgabe einer Identifikation der Ressourcen von RESTful SOA aus fachlichen Modellen ein. Eine genauere Untersuchung der Zusammenhänge zwischen Begrifflichkeiten und Konzepten der mit dem Anwendungssystem RESTful SOA unterstützten Aufgaben erfolgt kaum. Einen Lösungsbeitrag zur Verringerung der hierbei bestehenden semantischen Lücke liefert die SOM-R-Methodik der vorliegenden Arbeit.

8.1.2 Modellierung von REST-Anwendungen

Eng verbunden mit der modellgetriebenen Entwicklung von RESTful SOA ist die *Modellierung* ihrer *RESTful Services*. Eine Reihe wissenschaftlicher Beiträge schlägt Modellierungsansätze für RESTful WS und ihre Schnittstellen vor. Diese Arbeiten verfolgen insbesondere zwei Ziele:

- Bereitstellung von Modellierungssprachen für die modellgetriebene Ableitung und Beschreibung von implementierungsnahen REST-Spezifikationen (erstes Ziel)
- Etablierung von formalen und maschinen-lesbaren Spezifikationssprachen der REST-Schnittstelle (zweites Ziel)

Mit der Definition von **REST-Metamodellen** wird das Erreichen des erstgenannten Ziels angestrebt. Ein detailliertes Metamodell zur Darstellung von RESTful Anwendungen stammt von SCHREIER [Schr11]. Die RESTful Modellbildung unterstützen hier ein strukturorientiertes Metamodell zur Spezifikation der Ressourcenschnittstelle sowie ein verhaltensorientiertes Metamodell zur Beschreibung der Ausführungszustände von Ressourcen. Die Verhaltensspezifikation basiert dabei auf dem Konzept des Zustandsautomaten. Dieser Aspekt wird jedoch nicht tiefergehend vorgestellt. ALARCON und WILDE ([AlWi10a], [AlWi10b]) schlagen das Metamodell der Sprache ReLL (Resource Linking Language) vor. Ihr Fokus liegt auf der Beschreibung von Verlinkungen zwischen Ressourcen, um hieraus semantische Ressourcenspezifikationen mittels RDF (Resource Description Framework nach [W3C14]) abzuleiten. In der Literatur existiert darüber hinaus eine Reihe von Vorschlägen zur REST-Modellierung (z. B. [TaVa13], [VaPa09]).

Häufig stellen Modelle von REST-Services auf der softwaretechnischen Ebene den Ausgangspunkt für die Generierung von formalen, maschinen-lesbaren Spezifikationen der Ressourcenschnittstelle dar (zweites Ziel). Ihre Modellierung erfolgt mit **Service-Beschreibungssprachen**. Auf der Implementierungsebene reichen die hierbei eingesetzten Sprachen von XML-basierten Ansätzen, wie WADL (Web Application Description Language nach [Hadl09]) und WSDL 2.0 (Web Service Description Language nach [CMR+07]) der W3C, bis hin zu Spezifikationen auf Basis des JSON-Formats, wie JSON-LD [LaGü12] und RestDoc [Rest12], HTML, wie hREST [KGV08] oder der RDF, wie SA-REST ([SGL07], [LGS07]).

Zusammenfassend ist festzuhalten, dass in der Literatur eine Vielzahl von Modellierungssprachen bzw. Metamodellen zur Beschreibung der REST-Schnittstelle existiert. Mit dem Einsatz dieser Sprachen wird meist die Spezifikation von RESTful Services angestrebt, um aus den spezifizierten Modellen formale Schnittstellenbeschreibungen zu generieren. Eine integrierte Betrachtung der Außen- und Innensicht von RESTful SOA sowie die Einbeziehung von höheren Abstraktionsebenen in einem durchgängigen und einheitlichen Begriffssystem, wie sie die SOM-R-Methodik vorschlägt, sind in den vorgestellten Arbeiten nicht Gegenstand der Untersuchung. Beziehungen zwischen Modellebenen werden hier primär für die Ebenen der Softwarearchitektur und der Implementierung definiert (siehe auch Abschnitt 8.1.1).

8.1.3 Business Process Models und RESTful Web Services

Die Beziehung zwischen *REST* und *Business Processes* bzw. *Workflows*[72] wird in der aktuellen Forschung vor allem aus zwei Perspektiven untersucht. Einerseits erfolgt das Erstellen neuer, oder das Anpassen bestehender Workflowsprachen, um die *Komposition von RESTful Web Services* spezifizieren zu können (z. B. [Over08], [CDK+07]). Andererseits werden Mappings der *REST-Schnittstelle* auf die Bausteine von *Workflowsprachen* definiert (z. B. [XZL+08a]).

Die Arbeiten aus dem Themenbereich **RESTful Workflowsprachen** werden im Folgenden nach der verwendeten Spezifikationssprache zusammengefasst.

[72] Der englische Begriff des „Business Process“ wird in der Literatur häufig synonym zum Begriff des Workflows verwendet und beschreibt die Steuerung von Aktivitäten zur Durchführung einer fachlichen Aufgabe. Zur Abgrenzung vom Begriff „Geschäftsprozess“ wird in dieser Arbeit deshalb von Workflows gesprochen (vgl. dazu Abschnitt 4.1.2).

- Eine Erweiterung von *BPEL* um Modellelemente zur Beschreibung von Ausführungszuständen des AwS sowie der Komposition von RESTful Web Services schlagen OVERDICK [Over08] und PAUTASSO vor ([Paut08], [Paut09b]). In diesen Bereich fällt auch die BPEL-ähnliche Orchestrierungssprache *Bite* ([CDK+07], [RCD+08]).
- *Petri-Netz-basierte* REST-Kompositionssprachen beschreiben ALARCON und WILDE [AWB11] sowie DECKER ET AL (Service Nets, [DLO+09]).
- Mit *BPMN for REST* führt PAUTASSO eine Erweiterung von BPMN 2.0 ein und definiert Symbole zur Spezifikation von Ressourcen und der Interaktion mit diesen über HTTP-Verben [Paut11]. WOLF und BENKER verwenden textuelle Annotationen, um die Komposition von RESTful Services mittels BPMN zu beschreiben ([Wolf12], [WoBe13]).
- Weitere RESTful Workflowsprachen finden sich in den Arbeiten zu *JOpera* [Paut09a] sowie der Vorstellung eines *Situationskalküls* zur Komposition von RESTful Web Services [ZhDo09].

Die **RESTful Beschreibung von Workflows** („Business Processes“) bildet den zweiten zu untersuchenden Gegenstand. Im Fokus der Untersuchungen steht die Interpretation der konzeptuellen Bestandteile von Ablaufmodellen als Ressourcen, die über eine einheitliche Schnittstelle zugreifbar und manipulierbar sind. Beispielsweise werden Prozesse, Aktivitäten, Prozessinstanzen sowie deren Ausführungszustände in Form von Ressourcen beschrieben (z. B. [PaWi11], [XZL+08a], [XZL+08b]). Damit kann, ähnlich dem Ansatz der Spezifikation von VOTs als Vorgangsressourcen, die Ablauflogik eines Systems gekapselt und über Standard-Operationen verwaltet werden (vgl. Abschnitt 5.5.5.2).

In der vorliegenden Arbeit wird ebenfalls eine Erweiterung des BPMN-Metamodels vorgeschlagen (Abschnitt 5.3.6). Die Gründe für die Konzeption eines eigenen Ansatzes liegen vor allem in der Erreichung von zwei Gestaltungszielen. Erstens soll die Veröffentlichung von Vorgangsservices als RESTful Services und die Modellierung deren Innensicht als RESTful Workflowspezifikation konsistent auf das Begriffssystem der SOM-R-Methodik abgestimmt werden. Zweitens soll eine Erweiterung des Metamodells der BPMN derart erfolgen, dass ein erstelltes Workflowschema weiterhin auf die Syntaxanforderungen gängiger BPMN-WfMS anpassbar ist, und damit auch ausführbar bleibt. Die vorgenommenen Metamodellanpassungen beschränken sich deshalb auf die Spezialisierung bestehender Metaobjekte (REST-spezifische Aktivität) und die Zuordnung von Eigenschaften (Schnittstellenmerkmale und Nachrichtenparameter). Grundsätzlich ist auch eine Integration von verwandten Ansätzen aus der REST-Workflowmodellierung in das SOM-R-Architekturmodell denkbar (z. B. [Paut11]). Voraussetzungen dafür sind jedoch die Konstruktion eines Metamodells sowie dessen Abstimmung mit dem Begriffssystem der Entwicklungsmethodik.

Als abschließendes Fazit der bisherigen Literaturbetrachtung ist festhalten, dass in den beleuchteten Arbeiten eine modellbasierte Überführung der Konzepte von Geschäftsprozessmodellen (Aufgabenebene) in Komponenten von REST-Anwendungen nur lückenhaft untersucht werden. Auch werden in diesen Arbeiten keine Metamodelle für die RESTful Workflowsprachen definiert. Wie bereits in den vorausgegangenen Abschnitten aufgezeigt, steht primär die Lösung softwaretechnischer Probleme im Fokus der Betrachtung.

8.2 Ausgewählte SOA-Entwicklungsansätze

Die Untersuchungen des Abschnitts 8.1 zeigen Defizite in existierenden Lösungsansätzen bezüglich einer durchgängig modellbasierten bzw. modellgetriebenen Entwicklung von REST-Anwendungen ausgehend von nicht-technischen Spezifikationen. Nachfolgend wird der zu untersuchende Betrachtungsgegenstand auf verwandte Arbeiten aus dem Bereich der *Entwicklung von SOA auf Basis von Geschäftsprozessen* erweitert und diese der SOM-R-Methodik gegenübergestellt.

Die Analyse der Merkmale und die Klassifikation von SOA-Entwicklungsansätzen sind Gegenstand einer Reihe von Literaturbeiträgen (z. B. [GuLa10], [BKO08], [RDS07]). Grundsätzlich lassen sich die Entwicklungsansätze anhand ihrer hierarchischen Struktur in *Top-Down-*, *Meet-in-the-Middle-* und *Bottom-Up-*Ansätze unterscheiden. Die weiteren Ausführungen richten den Fokus auf Top-Down-Entwicklungsansätze bzw. -methoden, da diese SOA ausgehend von Anforderungen auf fachlicher Ebene spezifizieren und damit dem Vorgehen in der SOM-R-Methodik entsprechen. Eine modellgetriebene Entwicklung von SOA ist dabei häufig an der MDA ausgerichtet.

Die Untersuchung wird in zwei Schritten mit unterschiedlichen thematischen Schwerpunkten durchgeführt:

- Modellgetriebene Entwicklung von SOA mit der MDA (Abschnitt 8.2.1)
- Methodologien der SOA-Entwicklung (Abschnitt 8.2.2)

Zunächst werden die Charakteristika von ausgewählten Ansätzen bzw. Methoden genauer betrachtet und diese anschließend anhand ausgewählter Merkmale der SOM-R-Methodik gegenübergestellt.

8.2.1 Entwicklung von SOA auf Basis der MDA

Die MDA [OMG01] bildet den Rahmen für eine Vielzahl modellgetriebener SOA-Entwicklungsansätze (z. B. [DRG+10b], [GTP07], [HPF10], [RRS+06]). Als grundlegende Zielsetzung wird mit der MDA die Erzeugung von plattformspezifischen Implementierungen in einem durchgängigen und automatisierten Top-Down-Entwicklungsprozess verfolgt. Das dabei gebildete Modellsystem ist hierarchisch in die Modellebenen CIM, PIM und PSM untergliedert (Abschnitt 2.3.2). Die SOM-R-Methodik ist zwar nicht explizit nach dem Konzept des MDA-Architekturrahmens gestaltet, jedoch kann ausgehend von den hierarchischen Modellebenen und dem modellgetriebenen Vorgehen eine Gegenüberstellung und Diskussion ihrer Eigenschaften mit MDA-basierten Ansätzen erfolgen.

Zunächst erfolgt die Einschränkung der potenziell zu untersuchenden MDA-Ansätze durch die Bestimmung des **Zielsystems der Entwicklung** als *service-orientierte Architektur* (SOA). Die SOA wird ausgehend von der fachlichen Quelle *Geschäftsprozess* (Business Process, Workflow) *modellgetrieben* (oder modellbasiert) spezifiziert. Das Quellmodell (CIM, PIM) muss sich zudem auf der fachlichen Aufgabenträger- oder einer höheren Abstraktionsebene befinden.

Im Folgenden wird ein **Überblick** über verwandte MDA-Ansätze gegeben. Korrespondierend zu den Untersuchungsergebnissen der modellgetriebenen Entwicklung von REST-Schnittstellen lässt sich auch für den MDA-basierten Entwurf von SOA feststellen, dass eine Vielzahl von Beiträgen primär die Ableitung von Spezifikationen aus plattformnahen Beschreibungsebenen fokussiert. Dies gilt sowohl allgemein für den Bereich der modellgetriebenen SOA-Entwicklung (z. B. [RRS+06], [BBC+04], [YaPa04]), als auch im Speziellen für die modellgetriebene Ableitung aus Geschäftsprozess- bzw. Workflowmodellen (z. B. [MRS11], [DRG+10b], [TLD+07], [GTP07], [Rick07], [GTP07], [HPF10]). Generell ist eine Konzentration auf die Modellierung der Verhaltenssicht von Geschäftsprozessen und SOA feststellbar.

Nur wenige Beiträge betrachten die ganzheitliche Modellierung betrieblicher Informationssysteme und leiten fachliche AwS-Spezifikationen (PIM) aus Geschäftsprozessen (CIM) ab (z. B. [CMW09]) [TeSi12, S. 1664]. In diesen Arbeiten wird i. d. R. auf drei Abstraktionsebenen modelliert. Ausgehend von Geschäftsprozessmodellen, die z. B. in Form von EPK oder BPMN beschrieben sind (Ebene CIM), wird ein Zwischenmodell (PIM), das meist als BPMN-Schema spezifiziert ist, abgeleitet und manuell um technische Details angereichert. Abschließend erfolgt die Transformation in eine ausführbare Orchestrierungssprache (PSM) (z. B. [TLD+07], [GTP07], [Rick07]). Ausgehend von BPMN-Schemata leiten zudem eine Reihe von Beiträgen initiale Servicemodelle (PIM) ab, aus denen die Schnittstellenspezifikationen von Services (PSM) automatisiert erzeugbar sind (z. B. [HPF10], [DRG+10b]).

Für einen detaillierteren Vergleich mit der Methodik der vorliegenden Arbeit werden die *Service-Oriented Development Method* (SOD-M) ([CMW09], [CMV10]) und *MINERVA* ([DRG+10a], [DRG+10b], [DRG+09]) gewählt. Beide Ansätze verfolgen das Ziel der modellgetriebenen Entwicklung von SOA aus Geschäftsprozessmodellen auf der Ebene des CIM. Zudem treten sie, neben dem Aspekt der Modellierung des CIM, aus der Vielzahl an modellgetriebenen Ansätzen dadurch hervor, dass sie Vorgehen, Metamodelle und Transformationen der Entwicklung spezifizieren. Dies ist Voraussetzung für die Gegenüberstellung mit den Konzepten der SOM-R-Methodik und erhöht die Nachvollziehbarkeit der Untersuchung.

Der Untersuchung der ausgewählten Arbeiten werden vier Fragen an die Seite gestellt:

- Auf welchen *Abstraktionsebenen* bzw. Beschreibungsebenen der Systementwicklung erfolgt die Spezifikation des Modellsystems?
- Werden die *Merkmale des Modellsystems* vollständig beschrieben oder wird nur ein Teilsystem gebildet?
- Ist der *Prozess* der Modellbildung (auf den Ebenen des Modellsystems) *durchgängig* definiert und *automatisiert* durchführbar bzw. bis zu welchem Grad?
- Wird die *Weiterentwicklung* des Modellsystems untersucht bzw. welche Lösungsverfahren unterstützen eine Durchführung dieser Aufgabe?

DIE METHODE SOD-M

Die Methode SOD-M ([CMW09], [CMV10]) ist ein MDA-basierter Ansatz zur modellgetriebenen Entwicklung von SOA aus Geschäftsprozessmodellen. Hierbei wird eine ganzheitliche Perspektive auf das IS eingenommen, welche über die Fokussierung auf die technischen Ebenen von

PIM und PSM hinausgeht und die Erstellung von Geschäftsprozessmodellen auf der Ebene des CIM vorsieht. Abbildung 8.1 stellt den Entwicklungsprozess der Methode dar. Dieser wird nachfolgend erläutert.

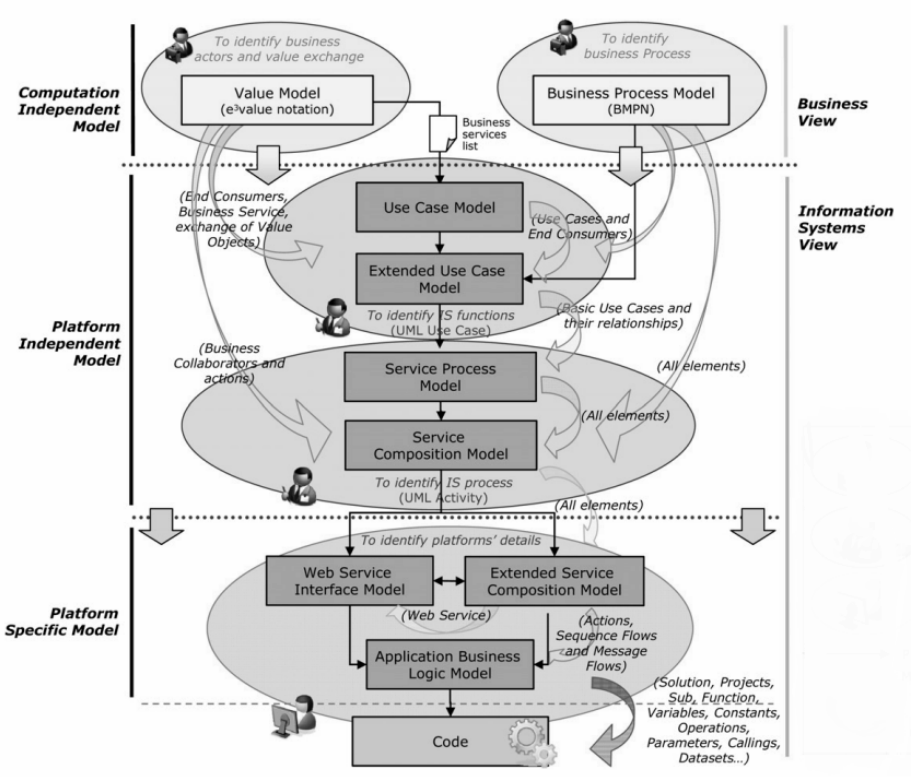


Abbildung 8.1: SOD-M Entwicklungsprozess [CMV10, S. 91]

Auf der Ebene CIM wird der Geschäftsprozess in Form von zwei Modellen beschrieben. Die Abläufe von durchzuführenden GP-Aufgaben werden in einem *BPMN-Schema* und die Leistungsflüsse zwischen den Prozessteilnehmer in einem Leistungsmodell mit der *e³value*-Notation [GoAk03] erfasst. Die Leistungen des Geschäftsprozesses werden dabei als Business Services interpretiert, die auf der Ebene des PIM jeweils einen *Use-Case* spezifizieren. Die Komposition und Ausführung der einzelnen Use-Cases wird in einem erweiterten Diagramm beschrieben. Die Modelle des CIM und das *erweiterte Use-Case-Diagramm* sind Ausgangspunkt für die Erstellung eines *Prozessmodells* (Service Process Model), welches den Ablauf der Aktivitäten von Business Services spezifiziert. Im *Kompositionsmodell* (Service Composition Model) wird anschließend der Workflow für die Serviceorchestrierung detailliert spezifiziert. Die Modellierung beider Servicemodelle basiert auf UML-Profilen des Aktivitätsdiagramms [CMV10, S. 90 ff.]. Die Spezifikation von *Web Service Interface Model*, *Extended Service Composition Model* und *Application Business Logic Model* im PSM erfolgt auf Basis der Sprachen WSDL und BPEL ([CMS06], [MAC06], [MCV03]). Die Modellierung der Schemata von CIM und PIM unterstützen Metamodelle der verwendeten Modellierungsansätze (e³value, BPMN) bzw. der erstellten UML-Profile (Use-Case- und Aktivitätsdiagramme).

Die Schritte zur Transformation von CIM in PIM sind als Mappingregeln definiert, die eine Überführung der CIM-Schemata auf Basis der Verknüpfung der, in den verwendeten Meta-modellen, gemeinsamen Konzepte *Business Task*, *Business Service*, *End Consumer*, *Business Collaborator* beschreiben. Der Automatisierungsgrad ist hierbei teilautomatisiert, da im Zuge

der Transformation Entwurfsentscheidungen getroffen werden müssen sowie die Detaillierung der spezifizierten Modelle notwendig ist [CMV10, S. 97]. Die Transformation des PIM in PSM sieht die Ableitung der Web-Service-Schnittstellen und des Kompositionsmodells vor, die abschließend in einem Anwendungsmodell zusammengeführt werden. Sie bilden die Basis für die Generierung der SOA-Implementierung.

Die Methode SOA-M wird anhand der eingangs vorgestellten Fragen reflektiert. So definiert die Methode einen MDA-basierten Prozess zur Spezifikation von SOA auf allen **Abstraktionsebenen** der Systementwicklung. Das CIM beschreibt den Geschäftsprozess aus Leistungs- und Ablaufsicht. Auf die Durchführung der Geschäftsprozesskoordination wird **keine strukturorientierte Perspektive** eingenommen, weshalb das Modellsystem auf der PIM-Ebene aus einer „primär" verhaltensorientierten Sicht spezifiziert ist. Folglich steht, entsprechend der vorliegenden Informationen der Quellmodelle, eine Identifikation und Modellierung von strukturellen Systemmerkmalen nicht im Fokus. Metamodelle unterstützen die Modellbildung auf den Ebenen. Integrierte Metamodelle werden für die einzelnen Modellebenen nicht vorgestellt.

Die **Transformation** von CIM in PIM erfolgt teilautomatisiert und definiert eine Menge von Abbildungsregeln sowie (personell durchzuführenden) Anreicherungsschritten. Die explizite Trennung der Transformation in einen automatisierten und nicht-automatisierten Teil, wie sie die Erstellung von initialen Schemata und deren Überarbeitung in der SOM-R-Methodik vorsieht, erfolgt nicht. Vielmehr werden Schemata des PIM häufig auf Basis von Transformationen innerhalb einer Modellebene erstellt, so dass die Modellbildung innerhalb des PIM als schrittweise Spezifikation von einzelnen Schemata realisiert ist. Beispielsweise erfordert die Ableitung des Service Composition Model die Erstellung des Use-Case-Model, des erweiterten Use-Case-Model und des darauf aufbauenden Service Process Model [CMV10, S. 91]. Ergebnis der PIM-Spezifikation ist das Service Composition Model, dessen Zielsetzung der des konsolidierten VOS ähnelt. Für die Unterstützung der Modellbildung von CIM und PIM wird ein Werkzeug auf Basis von EMF und ATL vorgestellt [CMV10, S. 102 f.].

Die Ableitung von PSM und dessen Implementierung konnte aufgrund fehlender Details zu ihrer Umsetzung nicht genauer analysiert werden. Ebenso sind keine Konzepte zur Unterstützung der **Weiterentwicklung** einer eingeführten SOA bei Änderungen in den zugrundeliegenden CIM-Schemata für die untersuchte Methode bekannt.

DAS MINERVA-FRAMEWORK

Als zweiter Ansatz zur modellgetriebenen Entwicklung von SOA wird Minerva vorgestellt ([DRG+10b], [DRG+10a], [DRG+11], [DWR+13]). Ausdrückliches Ziel des Frameworks ist die automatisierte Erzeugung von service-orientierten Systemen auf Basis von Geschäftsprozessmodellen. Zudem wird die Unterstützung der Phasen Analyse, Design, Konfiguration und Umsetzung (Realisierung) sowie Evaluation des Business-Process-Life-Cycles angestrebt. Abbildung 8.2 zeigt das schematische Vorgehen der Modellbildung und die zu erstellenden Modelle.

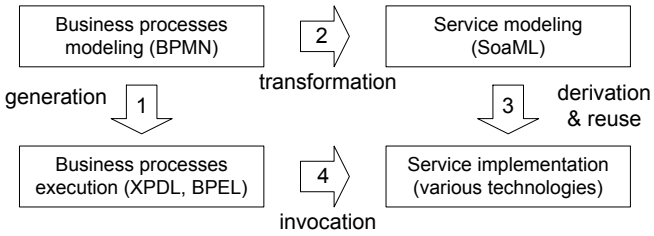


Abbildung 8.2: Beziehungen zwischen Geschäftsprozess und Services [DRG+10b, S. 458]

Den Ausgangspunkt der SOA-Entwicklung bilden Geschäftsprozessmodelle, die mit der BPMN spezifiziert sind. Die BPMN-Schemata werden (1) in einem automatisierten Generierungsschritt in ausführbare Workflowspezifikationen (BPEL oder XPDL[73]), sowie (2) in einem teilautomatisierten Transformationsschritt in Servicebeschreibungen auf Basis der SoaML [OMG12] überführt. Aus der SoaML-Spezifikation wird anschließend (3) ein plattformspezifisches Implementierungsmodell abgeleitet und (4) dieses mit den Aufrufbeziehungen des Workflows abgestimmt ([DRG+10a, S. 46], [DRG+10b, S. 457 f.]). Nach dem Ansatz der MDA lassen sich die BPMN-Schemata der Ebene CIM, die SoaML dem PIM und die Workflowspezifikation sowie die Implementierung dem PSM zuordnen [DRG+11, S. 677].

Entwicklungsaktivitäten legen die Schritte zur (teilautomatisierten) Durchführung der Serviceentwicklung fest. So erfolgt beispielsweise die Abstimmung von Workflows und der implementierten Serviceschnittstelle in einem nicht-automatisierten Schritt. Als Basis der Modellbildung und der Definition von Transformationen dienen die Metamodelle der eingesetzten Modellierungsansätze. Eine integrierte Werkzeugunterstützung ist auf Basis von Eclipse realisiert ([DRG+10a, S. 47 ff.], [DRG+11, S. 674 ff.]).

In Minerva werden Geschäftsprozesse aus einer **verhaltensorientierten Sicht** beschrieben. Die Leistungs- und Lenkungssicht des Geschäftsprozesses werden dagegen nicht explizit untersucht. Entsprechend fokussiert die SOA-Spezifikation auf das Verhalten und die Schnittstellenmerkmale von Web Services und nimmt **keine ganzheitliche Modellierung** des Zielsystems vor.

Als modellgetriebener Ansatz strebt das Framework bei der Entwicklung von Workflow, Servicebeschreibung und -implementierung einen **hohen Automatisierungsgrad** an. Um eine Generierung ausführbarer Workflowspezifikation realisieren zu können, erscheint jedoch eine detaillierte und aufgabenträgerspezifische Modellierung der Geschäftsprozesse notwendig. Dies rückt das Quellmodell auf die (fachliche) AwS-Ebene bzw. die PIM-Ebene. Das Erreichen einer hinreichenden Spezifikationstiefe der (initial) abgeleiteten SoaML-Schemata als Basis für die SOA-Implementierung erfordert deren Überarbeitung durch den Systementwickler. Die Anwendbarkeit des Ansatzes auf Modelle mit höherem **Abstraktionsgrad**, wie Geschäftsprozessmodelle auf Aufgabenebene, kann somit lediglich als eingeschränkt gegeben beurteilt werden.

Die **Überarbeitung** eines existierenden service-orientierten Systems adressiert das Framework in Zusammenhang mit der Umsetzung des „Business Process Life Cycles“ [DWR+13]. Als

[73] Die XML Process Definition Language (XPDL) [Work12] ist eine XML-basierte Sprache zur Beschreibung von Workflows. Sie wird von der Workflow Management Coalition (WfMC) standardisiert (Webseite: http://www.wfmc.org/standards/xpdl, Abruf am 11. Januar 2015).

Lösungsansatz wird ein Verbesserungsprozess vorgestellt, der die Evaluation bestehender Workflows, die Definition von Anpassungen, das Re-Design und die anschließende Generierung sowie Ausführung einer neuen Version beschreibt. Notwendigkeit und Anforderungen an das Anpassen von Workflows ergeben sich folglich aus der Messung der Prozessausführung und der Identifikation von Verbesserungspotenzialen im Rahmen der Evaluation. In einem modellgetriebenen Vorgehen werden die vorgeschlagenen Änderungen umgesetzt [DWR+13, S. 68 f.]. Der Ansatz verfolgt somit das Ziel der Optimierung bestehender Workflows und der korrespondierenden Servicenutzung. Flexible Geschäftsprozessmodelle sind demnach nicht der Auslöser der Systempflege. Auch erfolgt keine Betrachtung des entwickelten Gesamtsystems und seiner strukturellen Merkmale. Von einem detaillierten Lösungsverfahren zur Durchführung modellbasierter Systemanpassungen wird abstrahiert.

8.2.2 Etablierte SOA-Entwicklungsmethoden

Die Ausführungen des Abschnitts 8.2.1 zeigen, dass nur einzelne modellbasierte Ansätze neben der Aufgabe einer Neu-Entwicklung von Systemen auch deren Weiterentwicklung betrachten. Die Untersuchung verwandter SOA-Entwicklungsansätze wird deshalb abschließend auf allgemeine Methoden erweitert. Der modellbasierte bzw. modellgetriebene Charakter der Entwicklung steht nun nicht mehr im Vordergrund der Betrachtung. Im Folgenden liegt der Schwerpunkt auf verwandten Arbeiten etablierter SOA-Methoden, welche die Phasen der Pflege bzw. Wartung in den Entwicklungsprozess einschließen.

Einen **Überblick** über SOA-Entwicklungsmethoden gibt eine Reihe von Literaturanalysen (z. B. [GuLa10], [BKO08], [RDS07]). Nur wenige der existierenden Ansätze stellen eine umfassende Entwicklungsmethodik bereit, die den Lebenszyklus von SOA (weitestgehend) abdecken und alle Ebenen der Systementwicklungsaufgabe behandeln (z. B. [GuLa10, S. 40]). Exemplarisch werden die *Service-Oriented Modeling Approach* (SOMA) der IBM [Arsa04] sowie die *Mainstream SOA Methodology* (MSOAM) nach ERL [Erl08a], [Erl08c] als verbreitete und bekannte SOA-Entwicklungsmethoden herausgegriffen.

Die Entwicklung von SOA wird in SOMA und MSOAM phasenorientiert durchgeführt. SOMA ([Arsa04], [AGA+08]) definiert hierzu einen iterativen und inkrementellen Prozess. Durch die Integration von SOMA und dem Rational Unified Process (RUP) wird darüber hinaus eine schwergewichtige SOA-Methode spezifiziert [WAD+07]. MSOAM ist ein hersteller- und plattformunabhängiger Ansatz und leitet die Entwicklung von SOA in den verschiedenen Stadien ihres Lebenszyklus an [Erl08a, S. 358 ff.]. Im Fokus von MSOAM stehen dabei die Phasen der Analyse, Entwurf sowie Versionierung von Web Services. Als einzige, dem Verfasser der vorliegenden Arbeit bekannte, SOA-Entwicklungsmethode werden in einer Spezialisierung von MSOAM die Auswirkungen der Spezifikation von RESTful Services auf die einzelnen SOA-Entwicklungsphasen untersucht („MSOAM with REST“) [ECP+13, S. 129 ff.]. Aus diesem Grund wird MSOAM als verwandte Arbeit ausgewählt und im Weiteren genauer betrachtet.

Die **Mainstream SOA Methodology** definiert generische (Teil-)Prozesse zur Durchführung der Aufgaben in SOA-Entwicklungsprojekten. Das Ziel von MSOAM ist die Abdeckung des gesamten SOA-Lebenszyklus, wobei der Schwerpunkt auf den Phasen Analyse und Entwurf von Services liegt ([ECP+13, S. 129 ff.], [Erl08a, S. 375 ff.]). Mit einer Anpassung der Methode an die REST-

Prinzipien werden die drei Phasen auf die spezifischen Anforderungen einer Entwicklung von RESTful Services ausgerichtet [ECP+13, S. 129 f.]. Das Lösungsverfahren der Analyse, Entwurf und auch Versionierung von RESTful Services ist jeweils als Folge von generischen Schritten definiert. Zur Unterstützung der Durchführung der Phasen werden die allgemein zu berücksichtigenden Anforderungen untersucht sowie eine Vielzahl an Lösungsansätzen, Best Practices und Entwurfspattern vorgeschlagen (z. B. [ECP+13, S. 327 ff.]). Modelle dienen hierbei primär der Dokumentation von Anforderungen und der erarbeiteten Ergebnisse.
Mit der Zielsetzung, einen generischen Rahmen für die SOA-Entwicklung und die dabei zu durchlaufenden Phasen zu definieren, unterscheidet sich die MSOAM von der modellbasierten Systementwicklung in der SOM-R-Methodik. Aus diesem Grund beschränkt sich die nachfolgende Gegenüberstellung der beiden Methoden auf ausgewählte Merkmale.

Zunächst wird das Merkmal *modellbasierte Spezifikation* betrachtet. Allgemein sind die Phasen der Analyse, Gestaltung und Pflege von RESTful Services in MSOAM weniger formalisiert, während in der SOM-R-Methodik alle Beschreibungsebenen der Systementwicklung durchgängig modellbasiert spezifiziert werden. Eine Definition von Sichten oder der (integrierten) Metamodelle der zu modellierenden Ebenen ist in MSOAM nicht vorgesehen. Aufgrund der MSOAM-Zielsetzung werden die modellgetriebene Systementwicklung und damit auch die *Transformation* von Modellen nicht weiterführend thematisiert. Workflows dienen in MSOAM als Quelle von fachlichen Anforderungen an die Entwicklung von RESTful Services (z. B. [ECP+13, S. 151 ff.]). Dieses Vorgehen ist der Analyse von VOTs auf der fachlichen Aufgabenträgerebene und der Ableitung von Ressourcen in T2 ähnlich, wobei in MSOAM bereits eine detaillierte Workflowspezifikation vorliegt und keine Automatisierung der Entwicklungsschritte vorgenommen wird. Den Bereich der *Weiterentwicklung* adressiert MSOAM im Zusammenhang mit den Aufgaben „Monitoring“, „Maintenance“ und „Versioning“ von Services (z. B. [ECP+13, S. 343 ff.], [Erl08a, S. 358 ff.], [Erl08c]). Jedoch sind zur Durchführung dieser Aufgaben, im Besonderen der Systempflege, keine konkreten Lösungsverfahren spezifiziert. Im Gegensatz zur SOM-R-Methodik sind die Phasen der Analyse einer bestehenden AwS-Landschaft zur Projektvorbereitung, der SOA-Einführung sowie ihrer Administration und Versionierung Teil des MSOAM-Entwicklungsprozesses.

Zusammenfassend ist festzuhalten, dass sich MSOAM und die SOM-R-Methodik bezüglich angestrebter Zielsetzung und Reichweite unterscheiden. Beide Lösungsansätze decken unterschiedliche, sich jedoch teilweise ergänzende Aufgabenbereiche der SOA-Entwicklung ab und sind daher nur eingeschränkt vergleichbar. Aufgrund ihres durchgängig modellbasierten Vorgehens ist ein Einsatz der SOM-R-Methodik zur Realisierung der zentralen MSOAM-Entwicklungsphasen vorstellbar. Die SOM-R-Entwicklungsschritte zur Ableitung und Überarbeitung des REST-Architekturmodells könnten z. B. die (teilweise) Durchführung der Analyse- und Entwurfsphase in MSOAM modellbasiert beschreiben und diese um einen Prozess zur Durchführung der Implementierung ergänzen. Die Konsolidierung des REST-Architekturmodells könnte durch Entwurfspattern und Best Practices der MSOAM unterstützt werden.

8.3 Konzepte der modellgetriebenen Anpassung von Softwaresystemen

Die Ausführungen des Abschnittes 8.2.1 zeigen, dass die Phase der Systemwartung und damit auch die Weiterentwicklung von SOA in den untersuchten MDA-basierten Entwicklungsansätzen nicht bzw. nur vereinzelt thematisiert wird. Der Vision der MDE folgend, wird in diesen Arbeiten meist die Wartungsaufgabe auf der Abstraktionsebene der Quellmodelle durchgeführt. Somit werden Änderungen primär an den Quellmodellen oder auch dem eingesetzten Generatorwerkzeug vorgenommen. Eine neue, lauffähige und angepasste Version des Zielsystems würde bei einer konsequenten Umsetzung der MDE Vision somit automatisiert im Rahmen von Transformationen erzeugt (z. B. [SVE+07, S. 11 ff.], [PeMe06, S. 38], Abschnitt 2.3.1).

Das Ziel der vollständig automatisierten Systemerzeugung wird i. d. R. jedoch nicht erreicht, so dass meist Ergänzungen am generierten Code in einem manuellen Schritt vorzunehmen sind (z. B. [FrRu07, S. 39 f.], [PeMe06, S. 135 ff.]). Diese „manuellen“ Codeänderungen dürfen im Fall eines wiederholten Generatorlaufs jedoch nicht überschrieben werden ([SVE+07, S. 159 ff.], [PeMe06, S. 135]). Gängige Konzepte zur Unterstützung der Weiterentwicklung von Systemen im Rahmen einer sogenannten *Re-Generierung* lassen sich anhand der Modellebene strukturieren, auf der die Verwaltung der manuellen Spezifikationen erfolgt. Für die weitere Untersuchung werden die verwandten Arbeiten deshalb nach folgenden Zielen gruppiert:

- Erfassen von manuellen Änderungen auf Ebene des Quell- oder eines Zwischenmodells
- Schutz manueller Änderungen auf Ebene des Zielmodells
- Herstellen der Nachvollziehbarkeit von Überarbeitungen in Modellen (Traceability)

Erfassen manueller Modelländerungen auf höheren Abstraktionsebenen

Die Konzepte der vollständigen Re-Generierung einer Zielsystemspezifikation sind konform mit dem Erreichen der Vision von MDE oder auch MDA (Abschnitt 2.3.1). Dabei wird eine hinreichende Spezifikationstiefe der Quellmodelle und des eingesetzten Generators vorausgesetzt. Ist dies gegeben, so ist die Ableitung aller Implementierungsdetails des Zielsystems möglich. Die Überführung von Änderungen zwischen Quell- und Zielmodell beschränkt sich in einem solchen Fall auf die Durchführung der definierten Transformationen.

Normalerweise sind auf der Modellebene des Zielsystems jedoch manuelle Änderungen vorzunehmen. Die Lösungsansätze für diese Problematik beruhen meist auf der Etablierung eines Round Trip-Verfahrens zur Anreicherung des Quellmodells oder der Erfassung von nichtgeneriertem Code in separaten Modellen (z. B. [Fran03], [SVE+07, S. 45 f.]).

- *Round Trip-Verfahren* spiegeln Änderungen, die am generierten Code vorgenommen werden, wieder zurück in die Quellmodelle des Entwicklungsprozesses (Reverse Engineering) [SVE+07, S. 44 f.]. Bei der erneuten Generierung (Forward Engineering) müssen die manuellen Änderungen folglich nicht auf der Ebene des Zielmodells geschützt werden.
  Die Realisierung eines solchen Vorgehens erfordert jedoch die Bereitstellung von Sprachkonzepten auf der Abstraktionsebene der Quellmodelle, mit denen die Spezifikationsdetails des niedrigeren Abstraktionsgrades der Zielmodellebene gekapselt und vollständig

beschrieben werden können [Guld09, S. 6]. Beiträge aus diesem Themenbereich fokussieren häufig die direkte Transformation von Modell-zu-Code (z. B. [AnCz06], [SeKü04]).
Auch die Markierung von Bausteinen im Quellmodell, wie sie beispielsweise die MDA vorschlägt, würde in diesen Bereich fallen (Abschnitt 2.3.2). Einhergehend mit der Realisierung eines solchen „rückwärtigen" Vorgehens besteht jedoch die Gefahr, dass der Abstraktionsgrad im Quellmodell weit(er) abgesenkt wird.

- Das Konzept des Erfassens von manuell erstelltem Programmcode in einem separaten Modell liegt der zweiten Gruppe von Beiträgen zugrunde (z. B. [Guld09], [ALC08]). Die Umsetzung dieses Lösungsansatzes kann durch die Spezifikation weiterer Modelle parallel zum Quellmodellsystem oder auf einer Zwischenebene erfolgen, um den generierten Teil des Zielsystems separat erfassen zu können. Dieser kann somit vor der vollständigen Codegenerierung modellbasiert geändert werden [ALC08].
  Als Alternative dazu sei auch das Erfassen der manuell durchgeführten Schritte bzw. Operationen bei der Änderung der generierten Modelle mittels spezieller Transformationssprachen in einem Modifikationsmodell genannt [Guld09]. Die Anwendung dieses Vorgehens führt zur Erzeugung des Zielsystems in zwei Transformationen. Im Falle von Änderungen im Quellmodell werden die betroffenen manuellen Schritte der Transformation identifiziert und anschließend deren Anpassung auf die neue Version des generierten Zielmodells vorgenommen [Guld09, S. 8 f.].

Schutz manueller Anpassungen im Zielmodell

Die zweite Kategorie von untersuchten Beiträgen ist bestimmt durch das Ziel, den Schutz von manuellen Ergänzungen an einem erzeugten Zielmodell vor der Überschreibung durch einen erneuten Generierungslauf sicherzustellen. In der Literatur werden insbesondere die Verfahren der Definition von geschützten Bereichen (protected regions) sowie die getrennte Verwaltung von generiertem und nicht-generiertem Programmcode vorgeschlagen (z. B. [SVE+07, S. 46], [PeMe06, S. 135 ff.]).

- *Geschützte Bereiche* markieren den manuell erstellten Code eines Programms und schützen ihn vor Überschreibung. Bei der erneuten Generierung des Zielsystems werden die gekennzeichneten Bereiche ausgespart und bleiben damit unverändert. Problematisch in diesem Zusammenhang ist, dass sich der generierte und manuelle Code vermischen und so gemischte Artefakte bzw. Dateien entstehen ([SVE+07, S. 160], [PeMe06, S. 136 f.]).
- Der Ansatz einer sauberen *Trennung von manuellem und generiertem Code* basiert auf seiner Verwaltung in separaten Dateien. Die Realisierung dieses Konzepts kann bei Generierung von objektorientierten Programmiercode klassischerweise mittels Interfaces, abstrakter Klassen oder Entwurfsmustern erfolgen [SVE+07, S. 159]. Ein verbreiteter Ansatz ist die Verwendung einer mehrstufigen (häufig der dreistufigen) Vererbung. Dabei werden der generierte Code in Form von abstrakten Klassen und die manuellen Codeänderungen in den jeweils zugehörigen Implementierungsklassen verwaltet ([PeMe06, S. 137 f.], [SVE+07, S. 161]).

Die Beiträge des Re-Engineerings fokussieren insbesondere das Thema einer Abstimmung der plattformspezifischen Modellebene und dem daraus generierten Programmcode der Imple-

mentierung. Folglich bieten die vorgestellten Konzepte Lösungsverfahren für Probleme im Rahmen des dritten Transformationsschrittes T3 der SOM-R-Methodik an und können die konsistente Abstimmung zwischen REST-Architekturmodell und REST-Implementierung unterstützen. Die Lösungsansätze *mehrstufige Vererbung* oder auch *geschützte Bereiche* zur Bewahrung von manuellen Änderungen an Eigenschaften von Implementierungsbausteinen erscheinen hierfür praktikabel.
Da beide Ansätze auf der Zielmodellebene (Programmcode) realisiert werden, könnte sich eine Integration in die SOM-R-Methodik auf die Erweiterung oder Anpassung der Transformation T3 sowie des Metamodells der Zielarchitektur beschränken. Der Schwerpunkt beider Lösungen liegt auf dem Schutz der Eigenschaftsspezifikationen von Modell-Bausteinen. Eine Anwendung auf den Bereich der Überarbeitung von Modellstrukturen, wie sie im Rahmen der Konsolidierung von der AwS-Spezifikation und dem REST-Architekturmodell erfolgt, steht dagegen nicht im Fokus.

Auch das Konzept des *Round-Trip-Engineerings* zielt auf die Bewahrung von manuellen Änderungen am Programmcode und dient damit ebenfalls zur Unterstützung der Systempflege. Die Realisierung der vorgestellten Ansätze basiert jedoch im Kern auf der Anreicherung des Quellmodells, um damit die Änderungen am Zielmodell zu erfassen. Dieses Vorgehen würde in der SOM-R-Methodik zu einer Vermischung der verwendeten Modellierungsansätze sowie der spezifizierten Details führen und der hierarchischen Trennung von Abstraktionsgraden auf den Ebenen des Architekturmodells widersprechen. Dagegen ist die Erfassung von Änderungsschritten in einem separaten Modell konzeptionell mit dem Dokumentationsansatz dieser Arbeit verwandt, da diese Ansätze das gemeinsame Ziel der Nachverfolgbarkeit (Traceability) von Modellmanipulationen verfolgen (siehe nachfolgende Ausführungen).

MODEL-TRACEABILITY

Ansätze aus dem Bereich der *Model-Traceability* werden als letzte Gruppe in den verwandten Themenbereichen betrachtet. Im Fokus dieser Arbeiten steht die Verwaltung von Verknüpfungen (Traces) zwischen Modellen bzw. zwischen Modellen und dem daraus generierten Programmcode in „Traceability-Modellen“ [ANR+06, S. 515 f.]. Auf Basis der erfassten Traces können beispielsweise die Auswirkungen von Änderungen am Quellmodell auf die Bestandteile des jeweils zugehörigen Zielmodells (z. B. Programmcode) analysiert werden. Traces unterstützen zudem die Synchronisation von Quellmodell und Code bzw. Zielmodell (z. B. [OlOl07], [OlNe06]).

Die untersuchten Arbeiten zum Thema Traceability stammen primär aus den Bereichen der Codegenerierung sowie dem Requirements Engineering (z. B. [Rupp09], [Pohl08], [ANR+06], [OlNe06]). Grundsätzlich ist die Herstellung von Nachvollziehbarkeit in vielen wissenschaftlichen und praktischen Disziplinen von Interesse, in denen Modelle zum Zweck der Dokumentation oder als Gestaltungsartefakte eingesetzt werden. Für einen tiefergehenden Einstieg in bestehende Verfahren und auch Techniken aus dem Bereich der „Model Traceability“ sei auf die weiterführende Literatur verwiesen (z. B. [ANR+06] sowie [Dick02], [OlOl07], [CCY02], [Lete02], [MRP06]). Das Konzept der Traceability kommt in der vorliegenden Arbeit bei der Dokumentation struktureller Überarbeitungen und der Transformation von Schemabausteinen sowie der Erfassung von Beziehungen zwischen Modellelementen zum Einsatz (Abschnitt 6.3).

Der Traceability-Ansatz dieser Arbeit ist auf die Anforderungen der Systempflege und den methodischen Rahmen der SOM-R-Methodik abgestimmt.

Als Fazit der vorangegangenen Untersuchung ist festzuhalten, dass eine Vielzahl von Lösungsbeiträgen zur Bewältigung von Modelländerungen existiert. Diese fokussieren häufig jedoch die Behandlung von speziellen Problemstellungen bei der Re-Generierung von Programmcode. Konzepte aus dem Bereich der Model-Traceability bzw. der Dokumentation von Modelländerungen finden sich in dem in dieser Arbeit vorgeschlagenen Lösungsansatz zur Systempflege wieder. Verwandte Arbeiten, die, entsprechend Zielsetzung und Kontext der vorliegenden modellbasierten Entwicklungsmethodik, die Bewältigung von struktureller Flexibilität auch auf hohen Abstraktionsebenen in einem integrierten Lösungsansatz durchgängig unterstützen, sind nicht bekannt. Die untersuchten Beiträge dienen vielmehr als Grundlage für die Konzeption der einzelnen Bestandteile der Systempflege in dieser Arbeit.

8.4 Zusammenfassung

Die Untersuchung verwandter Arbeiten zeigt, dass in keinem der hierbei näher betrachteten Beiträge das aufgespannte Untersuchungsproblem dieser Arbeit methodisch adäquat behandelt wird. Im ersten Schritt werden Arbeiten beleuchtet, die die modellgetriebene Entwicklung von REST-Anwendungen, die Spezifikation der REST-Architekturebene sowie die RESTful Modellierung von Geschäftsprozessen behandeln. Anschließend werden Arbeiten zur modellgetriebenen Entwicklung von SOA aus Geschäftsprozessmodellen betrachtet. Für eine Gegenüberstellung mit der SOM-R-Methodik werden zwei MDA-Entwicklungsmethoden ausgewählt und ihre Eigenschaften im Detail untersucht. Zudem erfolgt ein Vergleich der SOM-R-Methodik mit einer Entwicklungsmethode für RESTful Services. Abschließend werden mit Lösungsansätzen zur Bewältigung von Modelländerungen verwandte Arbeiten aus dem Themenbereich der modellgetriebenen Anpassung von Softwaresystemen vorgestellt.

Allgemein lässt sich in MDA-basierten Ansätzen eine Fokussierung auf die plattformspezifischen Beschreibungsebenen der Systementwicklung feststellen. Nur vereinzelt wird hierbei die Aufgabenebene in die Modellbildung mit einbezogen. Zudem ist eine methodische Lücke in Bezug auf die ganzheitliche Betrachtung der Struktur- und Verhaltensmerkmale von Geschäftsprozessen sowie auf Lösungen zur Bewältigung von Änderungen auf höheren Abstraktionsebenen erkennbar. Die Weiterentwicklung von Systemen wird in diesen Arbeiten häufig nicht oder nur im Zusammenhang mit einer Re-Generierung behandelt.

Für die einzelnen Themenbereiche des vorliegenden Untersuchungsproblems existiert somit eine Reihe von Lösungsvorschlägen; jedoch ist im Kontext der SOM-R-Methodik kein durchgängiger modellbasierter Ansatz bekannt, der die ganzheitliche Spezifikation der Innen- und Außensicht von Informationssystemen behandelt. Den Mehrwert der vorliegenden Arbeit gegenüber den untersuchten verwandten Arbeiten bildet die Einnahme einer ganzheitlichen Sichtweise auf die vorgestellte Problemstellung sowie die Schaffung eines methodischen Rahmens für die modellbasierte Entwicklung und Anpassung von RESTful SOA an die Anforderungen strukturflexibler (SOM-)Geschäftsprozessmodelle.

9 Schlussbetrachtung

Die vorliegende Arbeit widmete sich der Entwicklung betrieblicher Anwendungssysteme und ihrer Anpassung an veränderte Anforderungen flexibler Geschäftsprozesse. Ausgehend von der Annahme, dass der Einsatz von RESTful SOA und der SOM-Methodik dazu beitragen kann, die Systementwicklung bei der Bewältigung dieser Herausforderung zu unterstützen, bestand das Untersuchungsziel in der Konstruktion einer Entwicklungsmethodik zur modellbasierten Gestaltung und Pflege von RESTful SOA auf Basis flexibler SOM-Geschäftsprozessmodelle. Den Untersuchungsgegenstand der Arbeit bilden die Spezifikation von RESTful SOA als Realisierungsform betrieblicher Anwendungssysteme sowie strukturflexible SOM-Geschäftsprozessmodelle, die den fachlichen Ausgangspunkt für die Durchführung des Entwicklungsprozesses beschreiben.

Die in dieser Forschungsarbeit erzielten Untersuchungsergebnisse werden im jeweils letzten Abschnitt der einzelnen Inhaltskapitel zusammengefasst (vgl. Abschnitte 2.4, 3.5, 4.4, 5.7, 6.8, 7.4 und 8.4). Zusätzlich findet in den Abschnitten 5.6 und 6.7 eine detaillierte Betrachtung der konzeptuellen Ergebnisse des erarbeiteten Lösungsansatzes *SOM-R-Methodik* statt.

In der nachfolgenden Schlussbetrachtung werden die Ergebnisse der Arbeit kritisch gewürdigt (Abschnitt 9.1). Die inhaltlichen Ausführungen enden mit einem Ausblick auf offene Fragestellungen, die weitere Forschungsaktivitäten in den behandelten Themenbereichen motivieren (Abschnitt 9.2).

9.1 Kritische Würdigung der Ergebnisse

Zusammenspiel von SOM und RESTful SOA

Das Übertragen der Semantik von Bausteinen aus Struktur- und Verhaltenssicht von SOM-Geschäftsprozessmodellen auf die Komponenten der RESTful SOA stellt ein wichtiges Ergebnis des konstruierten Lösungsansatzes dar. Von herauszuhebender Bedeutung ist hierbei, dass die Spezifikation von Ressourcen einer RESTful SOA sowie ihre Nutzung primär auf Basis der Begrifflichkeiten von Schemaobjekten der SOM-Geschäftsprozessebene erfolgt. Dieses Merkmal fördert die Behandlung der Systementwicklungsaufgabe im Rahmen der konzeptuellen Modellierung. Eine ausführliche Betrachtung der hieraus gewonnenen Erkenntnisse findet sich in Abschnitt 5.6.1.

Die Anleitung des Systementwicklers bei der Durchführung nicht-automatisierbarer Entwicklungsaufgaben sowie eine explizite Definition struktureller Überarbeitungsoperationen ermöglicht es sicherzustellen, dass die auf den Ebenen des SOM-R-Architekturmodells spezifizierten Elemente aus der Geschäftsprozessebene existenziell begründet werden. Dies ist die Voraussetzung dafür, dass die Auswirkungen von einzelnen Änderungen im strukturflexiblen SOM-Geschäftsprozessmodell auf die entwickelte RESTful SOA zu einem wesentlichen Teil bestimmbar sind. Eine zusammenfassende Erläuterung des konstruierten Ansatzes der Systempflege erfolgt in Abschnitt 6.7.1.

Die in der Einleitung formulierten Annahmen, dass die SOM-Methodik und RESTful SOA geeignet sind, die Komplexitätsbewältigung der System(weiter)entwicklung im Rahmen eines modellbasierten Ansatzes zu unterstützen, können damit bestätigt werden.

Erfüllung der aufgestellten Formalziele

Die SOM-R-Methodik unterstützt die Gestaltung und Pflege von betrieblichen Informationssystemen. Die an eine Lösung des Untersuchungsproblems dieser Arbeit gestellten Formalziele konnten in den Abschnitten 5.6.2 und 6.7.2 anhand des jeweiligen Stands des erarbeiteten Lösungsansatzes reflektiert und als erfüllt bewertet werden.

Dagegen wird nicht geprüft, ob bzw. zu welchem Grad ein Einsatz der SOM-R-Methodik die Effektivität und/oder Effizienz des modellbasierten SOA-Entwicklungsprozesses steigert. Im Hinblick auf diese offene Fragestellung sei jedoch angemerkt, dass sich die Methodik aus Sicht der konzeptuellen IS-Modellierung von verwandten Ansätzen durch die Bereitstellung eines umfassenden methodischen Architekturrahmens und einheitlichen Begriffssystems zur Förderung einer durchgängig modellbasierten Entwicklung abhebt (Kapitel 8). Des Weiteren demonstriert die prototypische Implementierung eines Softwarewerkzeugs die Realisierbarkeit von Automatisierungspotenzialen im Entwicklungsprozess (Kapitel 7). Diese genannten Aspekte verbessern die systematische und zielgerichtete Durchführung der Systementwicklung und erscheinen folglich geeignet, die Effektivität und Effizienz des Entwicklungsprozesses durch Einsatz des vorliegenden Lösungsansatzes ebenfalls positiv zu beeinflussen.

Reflexion der Anwendbarkeit der Methodik

Die praktische Anwendbarkeit der Entwicklungsmethodik wird in dieser Arbeit am Beispiel einer Fallstudie gezeigt (Abschnitte 5.5.8 und 6.6.5). Die modellgetriebene Entwicklung des Programmcodes der RESTful SOA, ausgehend vom SOM-Geschäftsprozessmodell der Fallstudie, demonstriert eine durchgängige Unterstützung des Systementwicklungsprozesses. Als Basis für die Ableitung des AwS-Programmcodes werden die Entwicklungsschritte zur Spezifikation der Implementierungsebene zudem am Beispiel zweier Basismaschinen konzipiert, was die Plattformunabhängigkeit des vorgeschlagenen Ansatzes bestätigt (Abschnitte 5.5.8.1 und B.3). Weiter veranschaulichen zwei Szenarien der Fallstudie, wie eine Behandlung von Änderungen in strukturflexiblen SOM-Geschäftsprozessmodellen durch die schrittweise und modellbasierte Anpassung der Abstraktionsebenen des entwickelten Informationssystems erfolgt.

Den Kern des Lösungsansatzes zur modellbasierten Pflege von Informationssystemen bilden die Hilfsmittel zur Unterstützung einer Bewältigung der strukturellen Änderung von Modellbausteinen. Nicht geklärt werden konnte in dieser Arbeit die Frage, ab welchem Umfang einer Geschäftsprozessmodelländerung sich eine vollständige Neu-Gestaltung des Modellsystems und damit der RESTful SOA empfiehlt. Zwar bestätigen die durchgeführten Änderungsszenarien eine Anwendbarkeit der SOM-R-Methodik für den Fall, dass sich die Modellanpassungen des strukturflexiblen SOM-Geschäftsprozessmodells auf die Kommunikationsbeziehungen zwischen zwei betrieblichen Objekten oder die einzelnen Bestandteile eines betrieblichen Objekts beschränken. Sind jedoch mehrere betriebliche Objekte von Anpassungen betroffen, so besteht die Gefahr, dass sich die Änderungen „fortpflanzen“. Folglich würden sich die Aus-

wirkungen einer Änderung auf eine Vielzahl von Bausteinen im Modellsystem erstrecken, die dann zu prüfen und ggf. anzupassen wären. Für die Bewertung des Aufwands zur Pflege eines bestehenden Systems oder dessen Neu-Gestaltung liegen bisher keine Kriterien vor. Die Systementwickler werden in diesem Zusammenhang bei einer Entscheidungsfindung somit nicht weiter unterstützt.

Die Anwendbarkeit der SOM-R-Methodik ist gemäß den Restriktionen der vorliegenden Arbeit beschränkt (Abschnitt 1.2.1). So wird bei der modellbasierten Spezifikation von RESTful SOA grundsätzlich eine Unterstützung des gesamten SOM-GPM durch das neu-entwickelte AwS vorausgesetzt. Der Entwurf einer partiellen SOA für den Teilbereich eines Geschäftsprozesses sowie die Herausforderung der Integration von RESTful SOA in eine bestehende Anwendungslandschaft werden nicht adressiert. In diesem Kontext erscheint die Erweiterung der SOM-R-Methodik um Ansätze zur Modellierung partieller SOA [TeSi12] ein gangbarer Lösungsweg.

Reflexion des Nutzens für Fachanwender und Entwickler

Die SOM-R-Methodik unterstützt die beteiligten Subjekte der Modellnutzung und -erstellung beim Erreichen der angestrebten Untersuchungsziele der Arbeit. Der Mehrwert der Methodik wurde bereits in den Zusammenfassungen der einzelnen Kapitel sowie der vorausgegangenen kritischen Würdigung herausgearbeitet. SOM-Geschäftsprozessmodelle dienen als Mittel zur Kommunikation mit dem interessierten Fachanwender bzw. Domänenexperten. Im Kontext dieser Arbeit liegt ein Fokus auf der Herstellung von Nachvollziehbarkeit bezüglich der Umsetzung von Anforderungen des Geschäftsprozesses durch die Komponenten des entwickelten AwS. Die konsistente Abstimmung der hierzu erstellten Modelle ist auf Basis dokumentierter Abhängigkeiten zwischen Diensten und Geschäftsprozessbausteinen prüfbar.

Zur Unterstützung der Kommunikation zwischen Systementwickler (Modellierer, Softwareentwickler) und Fachanwender steht ein einheitliches Begriffssystem zur Verfügung. Auf der Ebene der Softwarearchitektur wird der Systementwickler bei der Spezifikation von RESTful SOA und der anschließenden Programmerstellung durch den Lösungsansatz dieser Arbeit methodisch unterstützt. Der flexible Aufbau der SOM-R-Methodik erlaubt zudem deren Anpassung an projektspezifische Anforderungen der Entwicklungsaufgabe.

Die Nutzung der SOM-R-Methodik setzt eine Einarbeitung der beteiligten Personen in die verwendeten Modellierungsansätze voraus. Die Untergliederung des zu erstellenden Modellsystems in Ebenen mit unterschiedlichen Abstraktionsgraden unterstützt die Bewältigung der Komplexität der Modellierungsaufgabe, ermöglicht ihre Zerlegung und eine Aufteilung auf den jeweiligen Modellierer. Die fachliche, objektorientierte AwS-Spezifikation bildet hierbei ein wichtiges Hilfsmittel, um die Perspektiven der Geschäftsprozessmodellierung und der Spezifikation der RESTful SOA zu integrieren und eine gemeinsame Kommunikationsbasis zu schaffen.

9.2 Ausblick

Die SOM-R-Entwicklungsmethodik ist das Ergebnis der vorliegenden Untersuchung. Weitere offene konzeptuelle sowie praxisbezogene Fragestellungen, die sich für den behandelten Themenbereich dieser Arbeit ergeben haben, werden im Folgenden aufgegriffen.

Unternehmensziele und nicht-funktionale Anforderungen: Die modellbasierte Spezifikation von RESTful SOA basiert in der vorliegenden Arbeit auf den fachlichen Anforderungen der Struktur- und Verhaltensmerkmale von Geschäftsprozessmodellen. Das konzipierte SOM-R-Architekturmodell beschreibt den methodischen Rahmen der Entwicklung. Weiterer Forschungsbedarf lässt sich hinsichtlich der Integration von strategischen Informationen und Unternehmenszielen in die bestehende Architektur erkennen, wodurch eine explizite Berücksichtigung der Zielebene des Unternehmens bzw. betrieblichen Systems möglich wird. Ferner ergeben sich aus der Einbeziehung von unternehmerischen Sach- und Formalzielen weitere Gestaltungspotenziale hinsichtlich der Spezifikation von nicht-funktionalen Anforderungen. Deren modellbasierte Umsetzung stellt ein umfassendes Forschungsfeld aus dem konzeptuellen Themenbereich dieser Arbeit dar.

Einsatz alternativer Zielarchitekturen: RESTful SOA bilden die Zielarchitektur einer modellbasierten Systementwicklung mit der SOM-R-Methodik. Die erarbeiteten Lösungsansätze zur Gestaltung und Pflege betrieblicher Informationssysteme sind jedoch nicht grundsätzlich auf RESTful SOA beschränkt. Die Übertragbarkeit der Ansätze auf weitere Zielplattformen stellt eine sowohl konzeptuelle, als auch praxisbezogene Fragestellung dar, die bereits im Formalziel der Flexibilität adressiert wird. In diesem Zusammenhang ist zu prüfen, inwieweit die konstruierten Hilfsmittel ganz, teilweise oder gar nicht übernommen werden können. Die fachliche AwS-Spezifikation markiert hierbei die zentrale plattformunabhängige Anwendungsebene der Methodik und damit den Ausgangspunkt für die konzeptuelle Anbindung weiterer Zielarchitekturen. Als Merkmal für Architekturen, die passend für den Einsatz in der SOM-R-Methodik sind, könnte die fachliche Ausrichtung ihrer Komponenten herangezogen werden. Weisen die softwaretechnischen Architekturbausteine einen fachlichen Fokus auf, so scheinen sie für einen Einsatz im Rahmen der modellbasierten Entwicklung aus Geschäftsprozessen geeignet. Eine Übertragung von Ergebnissen bzw. eine Anpassung der SOM-R-Methodik an eine alternative Zielarchitektur wäre dann wahrscheinlich möglich.

Werkzeugunterstützung: Die Funktionsanforderungen an ein integriertes Modellierungswerkzeug werden durch die SOM-R-Methodik bestimmt. Auch wenn die Realisierbarkeit einer Werkzeugunterstützung im Rahmen dieser Arbeit anhand eines Prototyps demonstriert wird, so ist die hohe Komplexität, die mit der Entwicklung einer praxistauglichen Modellierungssoftware einhergeht, bereits erkennbar. Für die Unterstützung des Systementwicklers können weitere Potenziale durch die Realisierung von Multi-Views in der bereitgestellten grafischen Nutzeroberfläche des Werkzeugs erschlossen werden. Einen Beitrag hierzu liefert z. B. die Implementierung der SOM-Methodik mit ADOxx [BoSi13]. Die Umsetzung der erarbeiteten Anwendungslogik und einer nutzerfreundlichen Oberfläche des Entwicklungswerkzeugs beschreibt ein umfangreiches praxisbezogenes Themenfeld für weitere Arbeiten.

Die genannten offenen Punkte umrahmen zukünftige Themenbereiche, deren Behandlung in Entwicklung und Forschung einen wichtigen Beitrag liefern kann, um weitere Unterstützungspotenziale für die ganzheitliche Durchführung der Systementwicklungsaufgabe zu erschließen.

A Erläuternde Fallstudie

A.1 SOM-Geschäftsprozessmodell

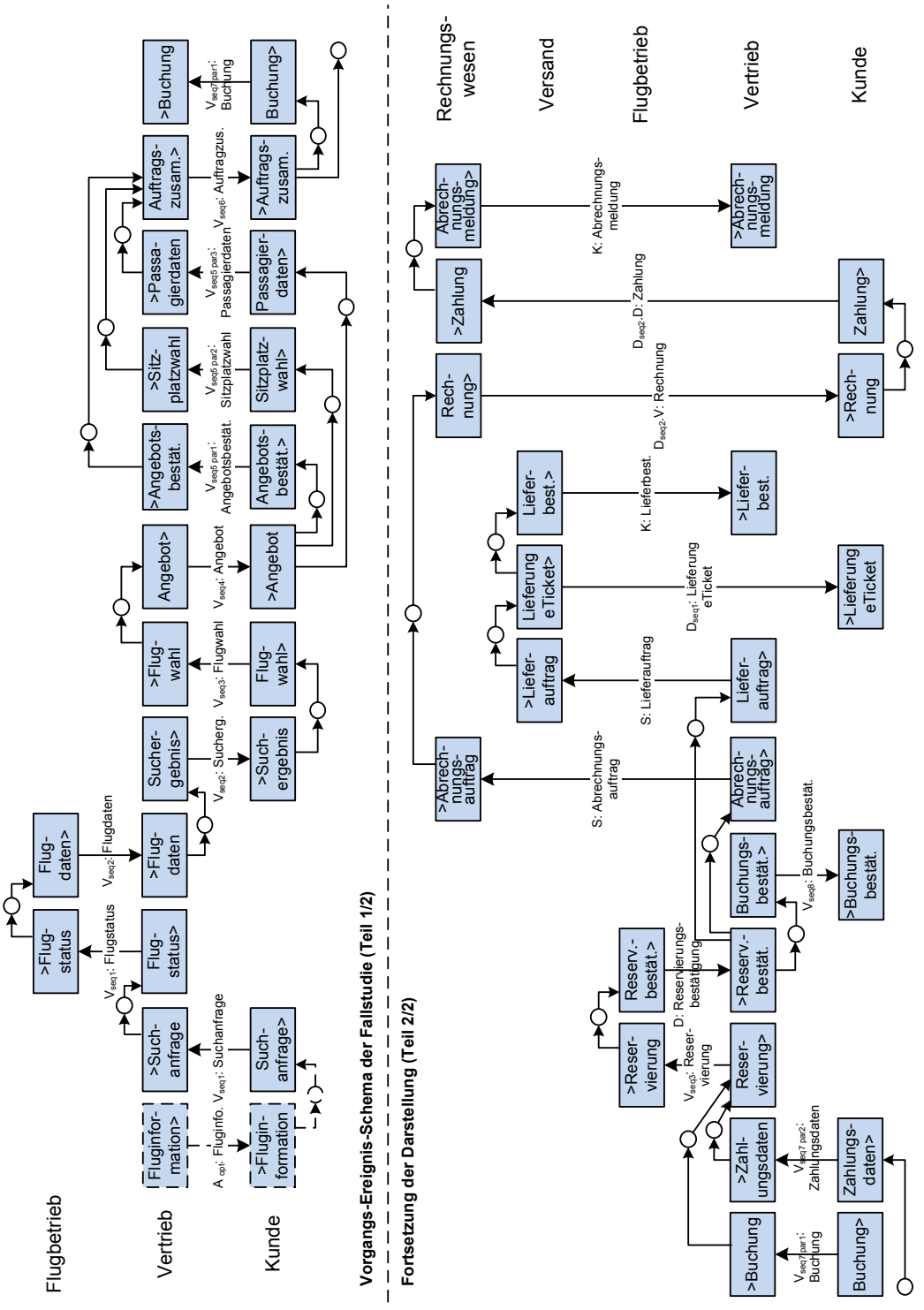


Abbildung A.1: Finales VES des SOM-Geschäftsprozessmodells$_1$ (Fallstudie)

A.2 Fachliche AwS-Spezifikation

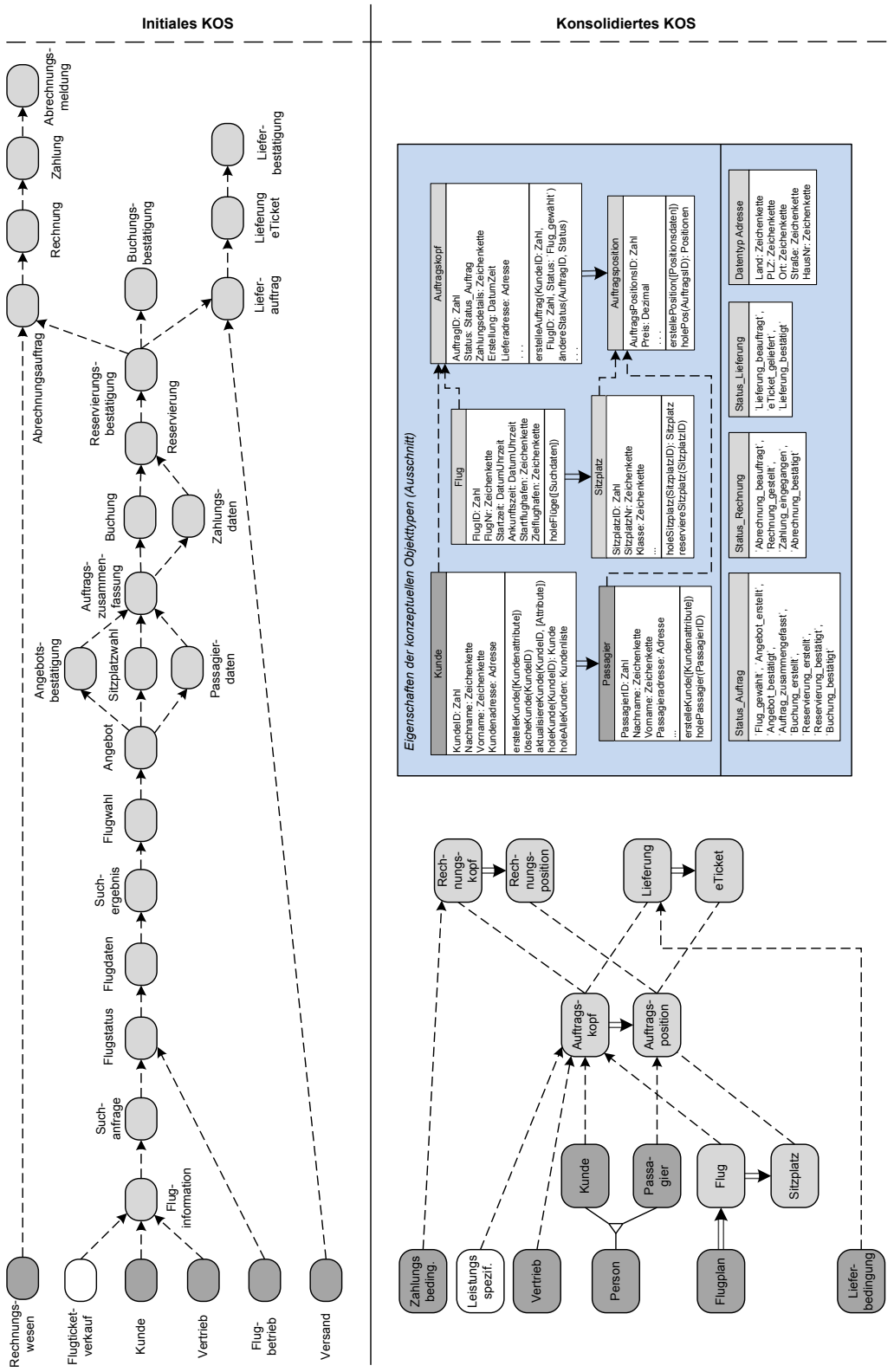


Abbildung A.2: Initiales KOS und konsolidiertes KOS der fachlichen AwS-Spezifikation$_1$ (Fallstudie)

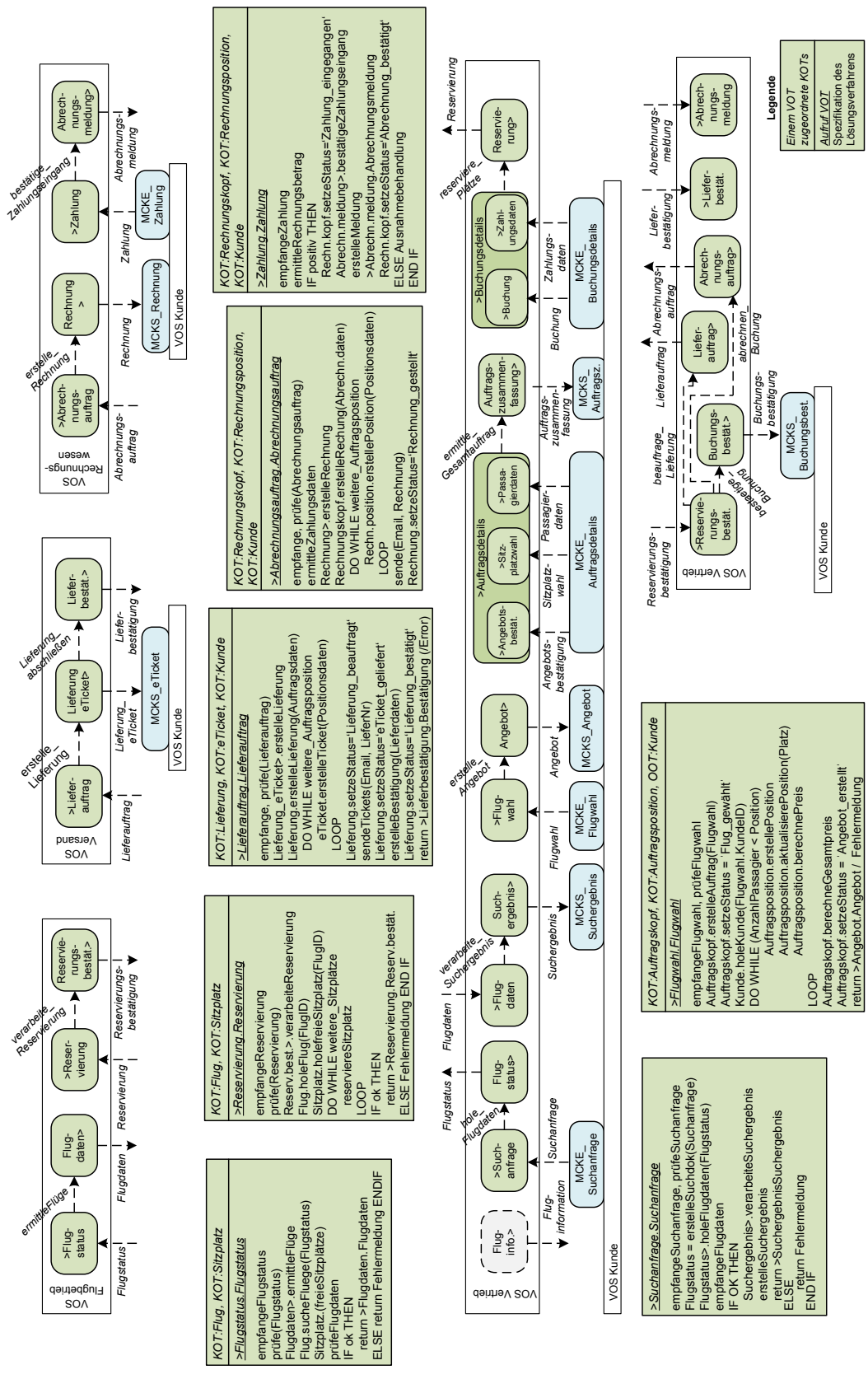


Abbildung A.3: Konsolidiertes VOS der fachlichen AwS-Spezifikation$_1$ (Fallstudie)

A.3 REST-Architekturmodell

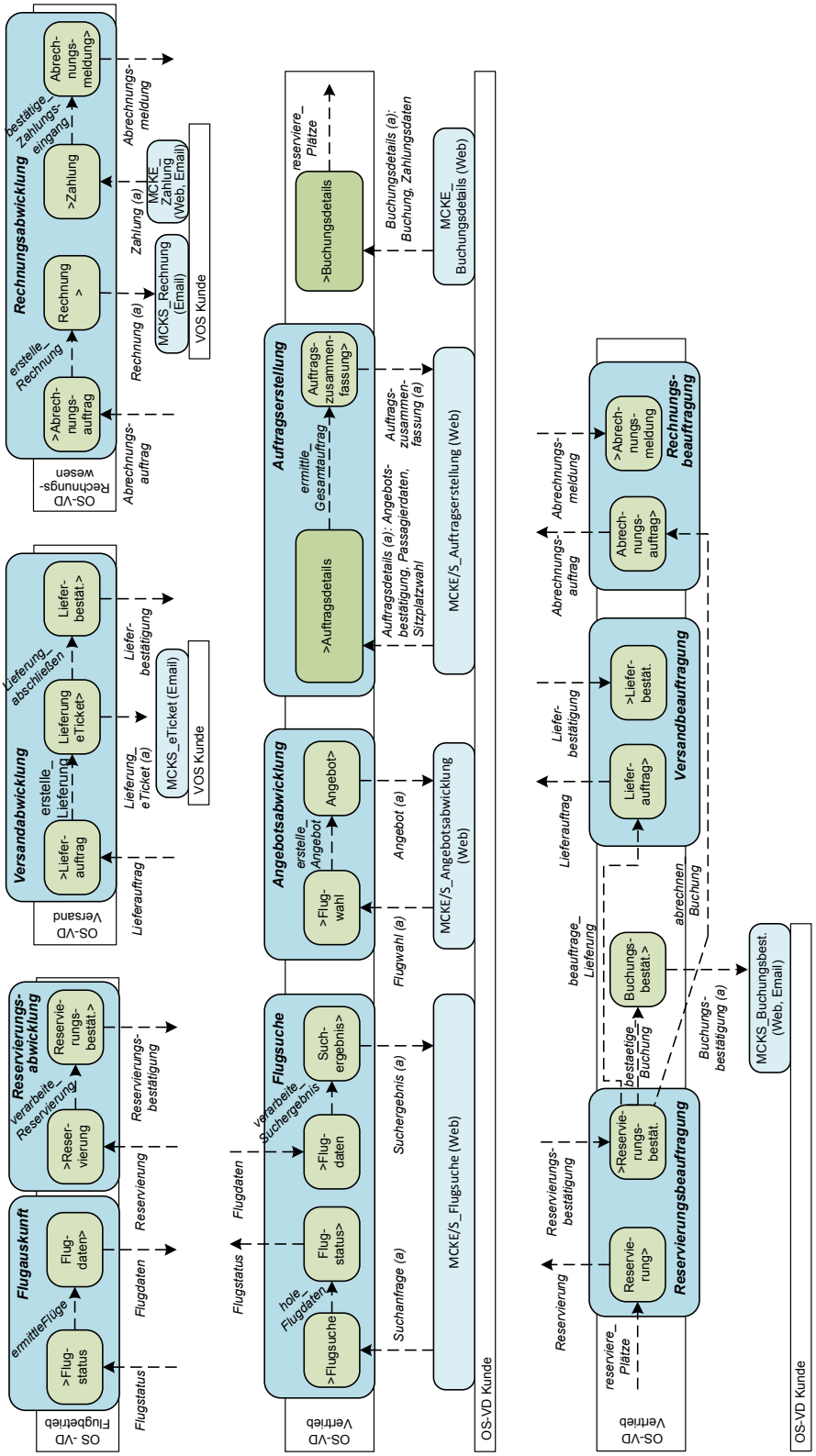


Abbildung A.4: Struktur des konsolidierten OS-VD des REST-Architekturmodells$_1$ (Fallstudie)

A.4 Dokumentation der Entwicklungsschritte T1 und K1

Nr	V	B	N	A[74]	I
1	bO:Vertrieb	⇒	OOT:Vertrieb	1	
2	bO:Flugbetrieb	⇒	OOT:Flugbetrieb	1	
3	bO:Versand	⇒	OOT:Versand	1	
4	bO:Rechnungswesen	⇒	OOT:Rechnungswesen	1	
5	bO:Kunde	⇒	OOT:Kunde	1	
6	L:Flugticketverkauf	⇒	LOT:Flugticketverkauf	1	
7	bT:A:Fluginformation	⇒	TOT:Fluginformation	1	
9	bT:V:Suchanfrage	⇒	TOT:Suchanfrage	1	
10	bT:V:Flugstatus	⇒	TOT:Flugstatus	1	
11	bT:V:Flugdaten	⇒	TOT:Flugdaten	1	
12	bT:V:Suchergebnis	⇒	TOT:Suchergebnis	1	
13	bT:V:Flugwahl	⇒	TOT:Flugwahl	1	
14	bT:V:Angebot	⇒	TOT:Angebot	1	
15	bT:V:Angebotsbestätigung	⇒	TOT:Angebotsbestätigung	1	
16	bT:V:Sitzplatzwahl	⇒	TOT:Sitzplatzwahl	1	
17	bT:V:Passagierdaten	⇒	TOT:Passagierdaten	1	
18	bT:V:Auftragszusammenfassung	⇒	TOT:Auftragszusammenfassung	1	
19	bT:V:Buchung	⇒	TOT:Buchung	1	
20	bT:V:Zahlungsdaten	⇒	TOT:Zahlungsdaten	1	
21	bT:V:Reservierung	⇒	TOT:Reservierung	1	
22	bT:D:Reservierungsbestätigung	⇒	TOT:Reservierungsbestätigung	1	
23	bT:V:Buchungsbestätigung	⇒	TOT:Buchungsbestätigung	1	
24	bT:S:Lieferauftrag	⇒	TOT:Lieferauftrag	1	
25	bT:D:Lieferung eTicket	⇒	TOT:Lieferung eTicket	1	
26	bT:K:Lieferbestätigung	⇒	TOT:Lieferbestätigung	1	
27	bT:S:Abrechnungsauftrag	⇒	TOT:Abrechnungsauftrag	1	
28	bT:D:Rechnung	⇒	TOT:Rechnung	1	
29	bT:D:Zahlung	⇒	TOT:Zahlung	1	
30	bT:K:Abrechnungsmeldung	⇒	TOT:Abrechnungsmeldung	1	

Tabelle A.1: Dokumentation von T1 der Fallstudie (IAS zu initialem KOS)

Nr	V	B	N
31	bA:A:Fluginformation>	⇒	VOT:Fluginformation>
32	bA:V:>Suchanfrage	⇒	VOT:>Suchanfrage
33	bA:V:Flugstatus>	⇒	VOT:Flugstatus>
34	bA:V:>Flugstatus	⇒	VOT:>Flugstatus
35	bA:V:Flugdaten>	⇒	VOT:Flugdaten>
36	bA:V:>Flugdaten	⇒	VOT:>Flugdaten
37	bA:V:Suchergebnis>	⇒	VOT:Suchergebnis>
38	bA:V:>Flugwahl	⇒	VOT:>Flugwahl
39	bA:V:Angebot>	⇒	VOT:Angebot>
40	bA:V:>Angebotsbestätigung	⇒	VOT:>Angebotsbestätigung
41	bA:V:>Sitzplatzwahl	⇒	VOT:>Sitzplatzwahl
42	bA:V:>Passagierdaten	⇒	VOT:>Passagierdaten
43	bA:V:Auftragszusammenfassung>	⇒	VOT:Auftragszusammenfassung>
44	bA:V:>Buchung	⇒	VOT:>Buchung
45	bA:V:>Zahlungsdaten	⇒	VOT:>Zahlungsdaten

[74] Auf die Angabe der Version 1 als Aktivitätszeitpunkt (A) wird in den weiteren Tabellen verzichtet.

46	bA:V:Reservierung>	⇒	VOT:Reservierung>
47	bA:V:>Reservierung	⇒	VOT:>Reservierung
48	bA:D:Reservierungsbestätigung>	⇒	VOT:Reservierungsbestätigung>
49	bA:D:>Reservierungsbestätigung	⇒	VOT:>Reservierungsbestätigung
50	bA:V:Buchungsbestätigung>	⇒	VOT:Buchungsbestätigung>
51	bA:S:Abrechnungsauftrag>	⇒	VOT:Abrechnungsauftrag>
52	bA:S:>Abrechnungsauftrag	⇒	VOT:>Abrechnungsauftrag
53	bA:D:Rechnung>	⇒	VOT:Rechnung>
54	bA:D:>Zahlung	⇒	VOT:>Zahlung
55	bA:K:Abrechnungsmeldung>	⇒	VOT:Abrechnungsmeldung>
56	bA:K:>Abrechnungsmeldung	⇒	VOT:>Abrechnungsmeldung
57	bA:S:Lieferauftrag>	⇒	VOT:Lieferauftrag>
58	bA:S:>Lieferauftrag	⇒	VOT:>Lieferauftrag
59	bA:D:Lieferung eTicket>	⇒	VOT:Lieferung eTicket>
60	bA:K:Lieferbestätigung>	⇒	VOT:Lieferbestätigung>
61	bA:K:>Lieferbestätigung	⇒	VOT:>Lieferbestätigung
62	bA:(bO:Kunde)	⇒	VOT:(VOS:Kunde)
63	bT:A:Fluginformation	⇒	Interacts_with(Fluginformation>,>Fluginfo.)
64	bT:V:Suchanfrage	⇒	Interacts_with(Suchanfrage>,>Suchanfrage)
65	bT:V:Flugstatus	⇒	Interacts_with(Flugstatus>,>Flugstatus)
66	bT:V:Flugdaten	⇒	Interacts_with(Flugdaten>,>Flugdaten)
67	bT:V:Suchergebnis	⇒	Interacts_with(Suchergebnis>,>Suchergebnis)
68	bT:V:Flugwahl	⇒	Interacts_with(Flugwahl>,>Flugwahl)
69	bT:V:Angebot	⇒	Interacts_with(Angebot>,>Angebot)
70-81 betriebliche Transaktionen …		⇒	Interacts_with(Aufgabe>,>Aufgabe)
82	bT:D:Zahlung	⇒	Interacts_with(Zahlung>,>Zahlung)
83	bT:K:Abrechnungsmeldung	⇒	Interacts_with(Abrechnungsmeldung>,>Abr.m.)

Tabelle A.2: Dokumentation von T1 der Fallstudie (VES zu initialem VOS)

Nr	V	B	N
	Überarbeitungsschritte 1 bis 5		
84	OOT:Vertrieb	→	OOT:Vertrieb (--- *keine Änderung* ---)
85	OOT:Flugbetrieb	→	OOT:Flugplan (--- *Umbenennung* ---)
86	OOT:Versand	→	OOT:Lieferbedingung (--- *Umbenennung* ---)
87	OOT:Rechnungswesen	→	OOT:Zahlungsbedingung (--- *Umbenennung* ---)
88	OOT:Kunde	→	OOT:Kunde (--- *keine Änderung* ---)
89	LOT:Flugticketverkauf	→	LOT:Leistungsspezifikation (--- *Umbenennung* ---)
90	TOT:Fluginformation	→	~~TOT:Fluginformation~~ --- *nicht automatisiert* ---
91	TOT:Suchanfrage	→	~~TOT:Suchanfrage~~ --- *transient* ---
92	TOT:Flugstatus	→	~~TOT:Flugstatus~~ --- *transient* ---
93	TOT:Flugdaten	→	TOT:Flug, TOT:Sitzplatz
94	TOT:Suchergebnis	→	~~TOT:Suchergebnis~~ --- *transient* ---
95	TOT:Flugwahl	→	TOT:Wahlkopf, TOT:Wahlposition
96	TOT:Angebot	→	TOT:Angebotskopf, TOT:Angebotsposition
97	TOT:Angebotsbestätigung	→	TOT:Angebotsbestätigungskopf, TOT:Angebotsbestätigungsposition
98	TOT:Sitzplatzwahl	→	TOT:Sitzplatzwahl (--- *keine Änderung* ---)
99	TOT:Passagierdaten	→	TOT:Passagier (--- *Umbenennung* ---)
100	TOT:Auftragszusammenfassung	→	TOT:Auftragszusammenfassungskopf, TOT:Auftragszusammenfassungsposition
101	TOT:Buchung	→	TOT:Buchungskopf, TOT:Buchungsposition
102	TOT:Zahlungsdaten	→	TOT:Zahlungsdaten (--- *keine Änderung* ---)

103	TOT:Reservierung	→	TOT:Reservierungskopf, TOT:Reservierungsposition
104	TOT:Reservierungsbestätigung	→	TOT:Reservierungsbestätigungskopf, TOT:Reservierungsbestätigungsposition
105	TOT:Buchungsbestätigung	→	TOT:Buch.bestätigungskopf, TOT:B.bestätigungspos.
106	TOT:Lieferauftrag	→	TOT:Lieferkopf, TOT:Lieferposition
107	TOT:Lieferung eTicket	→	TOT:Flugticket (--- *Umbenennung* ---)
108	TOT:Lieferbestätigung	→	TOT:Lieferbestätigung (--- *keine Änderung* ---)
109	TOT:Abrechnungsauftrag	→	TOT:Abrechnungskopf, TOT:Abrechnungsposition
110	TOT:Rechnung	→	TOT:Rechnungskopf, TOT:Rechnungsposition
111	TOT:Zahlung	→	TOT:Zahlung (--- *keine Änderung* ---)
112	TOT:Abrechnungsmeldung	→	TOT:Abrechnungsmeldung (--- *keine Änderung* ---)
113	OOT:Kunde, TOT:Passagier	→	KOT:Person, OOT:Kunde, TOT:Passagier
	Überarbeitungsschritte 6		
114	TOT:Wahlkopf, TOT:Angebotskopf, TOT:Angebotsbestät.kopf, TOT:Auftragszusammenf.kopf, TOT:Buchungskopf, TOT:Reservierungskopf, TOT:Res.bestätigungskopf, TOT:Buch.bestätigungskopf,	→	KOT:Auftragskopf
115	TOT:Wahlposition, TOT:Angebotsposition, TOT:Angebotsbestätigungspos., TOT:Sitzplatzwahl, TOT:Auftragszusammenf.pos., TOT:Buchungsposition, TOT:Reservierungsposition, TOT:Reservierungsbestät.pos., TOT:Buchungsbestät.pos.	→	KOT:Auftragsposition
116	TOT:Lieferkopf, TOT:K:Lieferbestätigung	→	KOT:Lieferung
117	TOT:Flugticket, TOT:Lieferposition	→	KOT:eTicket
118	TOT:Abrechnungskopf, TOT:Rechnungskopf, TOT:D:Zahlung, TOT:K:Abrechnungsmeldung	→	KOT:Rechnungskopf
119	TOT:Rechnungsposition , TOT:Abrechnungsposition	→	KOT:Rechnungsposition

Tabelle A.3: Dokumentation von K1.1 der Fallstudie (Überarbeitung KOS)

Nr	V	B	N
	Überarbeitungsschritte 1 bis 9 (zusammenfassend dargestellt)		
120	VOT:Fluginformation>	→	~~VOT:Fluginformation>~~ (Schritt 1)
121-136 VOT(VOS Kunde)		→	~~VOT:(VOS Kunde)~~
137	VOT:>Suchanfrage	→	VOT:>Suchanfrage, IOT:MCKE_Suchanfrage (Schritt 6.1)
138	VOT:Flugstatus>	→	VOT:Flugstatus> (--- *keine Änderung* ---)
139	VOT:>Flugdaten	→	VOT:>Flugdaten (--- *keine Änderung* ---)
140	VOT:Suchergebnis>	→	VOT:Suchergebnis>, IOT:MCKS_Suchergebnis
141	VOT:>Flugwahl	→	VOT:>Flugwahl, IOT:MCKE_Flugwahl
142	VOT:Angebot>	→	VOT:Angebot>, IOT:MCKS_Angebot
143	VOT:>Angebotsbestätigung, VOT:>Sitzplatzwahl VOT:>Passagierdaten	→	VOT:>Auftragsdetails (Schritt 5), IOT:MCKE_Auftragsdetails (Schritt 6.1)

144	VOT:Auftragszusammenf.>	→	VOT:Auftragszusammenf.>, IOT:MCKS_Auftragszus.f.
145	VOT:>Buchung VOT:>Zahlungsdaten	→	VOT:>Buchungsdetails IOT:MCKE_Buchungsdetails
146	VOT:>Zahlungsdaten	→	VOT:>Zahlungsdaten, IOT:MCKE_Zahlungsdaten
147	VOT:Reservierung>	→	VOT:Reservierung> (--- *keine Änderung* ---)
148	VOT:>Reservierungsbestät.	→	VOT:>Reservierungsbestät. (--- *keine Änderung* ---)
149	VOT:Buchungsbestätigung>	→	VOT:Buchungsbest.>, IOT:MCKS_Buchungsbest.
150	VOT:Lieferauftag>	→	VOT:Lieferauftag> (--- *keine Änderung* ---)
151	VOT:Abrechnungsauftrag>	→	VOT:Abrechnungsauftrag> (--- *keine Änderung* ---)
152	VOT:>Lieferbestätigung	→	VOT:>Lieferbestätigung (--- *keine Änderung* ---)
153	VOT:>Abrechnungsmeldung	→	VOT:>Abrechnungsmeldung (--- *keine Änderung* ---)
154	VOT:>Lieferauftrag	→	VOT:>Lieferauftrag (--- *keine Änderung* ---)
155	VOT:Lieferung eTicktet>	→	VOT:Lieferung eTicket>, IOT:MCKS_Liefer.eTicket
160	VOT:Lieferbestätigung>	→	VOT:Lieferbestätigung> (--- *keine Änderung* ---)
161	VOT:>Abrechnungsauftrag	→	VOT:>Abrechnungsauftrag (--- *keine Änderung* ---)
162	VOT:Rechnung>	→	VOT:Rechnung>, IOT:MCKS_Rechnung
163	VOT:>Zahlung	→	VOT:Zahlung>, IOT:MCKE_Zahlung
164	VOT:Abrechnungsmeldung>	→	VOT:Abrechnungsmeldung> (--- *keine Änderung* ---)

Tabelle A.4: Dokumentation von K1.2 der Fallstudie (Überarbeitung VOS/IOS)

A.5 Dokumentation der Entwicklungsschritte T2 und K2

Nr	V	B	N
	KOS zu OS-ED und Ressourcenschema (ER)		
165	OOT:Vertrieb	⇒	EOT:Vertrieb, ER: Vertrieb, ER:Vertriebsliste, DokT:Vertrieb
166	OOT:Flugplan	⇒	EOT:Flugplan, ER:Flugplan, ER:Flugplanliste, DokT:Flugplan
167	OOT:Lieferbedingung	⇒	EOT:Lieferbedingung, ER:Lieferbedingung, ER:Lieferbedingungsliste, DokT:Lieferbedingung
168	OOT:Zahlungsbedingung	⇒	EOT:Zahlungsbedingung, ER:Zahlungsbedingung, ER:Zahlungsbed.liste, DokT:Zahlungsbedingung
169	LOT:Leistungsspezifikation	⇒	EOT:Leistungsspezifikation, ER:Leistungsspezifikation, ER:Leistungsspez.liste, DokT:Leistungsspezifikation
170	KOT:Person	⇒	EOT:Person, ER:Person, ER:Personenliste, DokT:Person
171	OOT:Kunde	⇒	EOT:Kunde, ER:Kunde, ER:Kundenliste, DokT:Kunde
172	TOT:Passagier	⇒	EOT:Passagier, ER:Passagier, ER:Passagierliste, DokT:Passagier
173	KOT:Auftragskopf	⇒	EOT:Auftragskopf, ER:Auftragskopf, ER:Auftragskopfliste
174	KOT:Auftragsposition	⇒	EOT:Auftragsposition, ER:Auftragsposition, ER:Auftragspositionliste
175	KOT:Lieferung	⇒	EOT:Lieferung, ER:Lieferung, ER:Lieferungsliste
176	KOT:eTicket	⇒	EOT:eTicket, ER:eTicket, ER:eTicketliste
177	KOT:Rechnungskopf	⇒	EOT:Rechnungskopf, ER:Rechnungskopf, ER:Rechnungskopfliste
178	KOT:Rechnungsposition	⇒	EOT:Rechnungsposition, ER:Rechnungsposition, ER:Rechnungspositionsliste
179	TOT:Flug	⇒	EOT:Flug, ER:Flug, ER:Flugliste
180	TOT:Sitzplatz	⇒	EOT:Sitzplatz, ER:Sitzplatz, ER:Sitzplatzliste
181	Interacts_with(OOT:Vertrieb, KOT:Auftragskopf)	⇒	Interacts_with(EOT:Vertrieb,EOT:Auftragskopf)

182	ls_part_of(KOT:Auftragskopf, KOT:Auftragsposition)	⇒	Aggregation(EOT:Auftragskopf,EOT:Auftragsposition)
183-198 Beziehungen(KOS)		⇒	Beziehungen(OS-ED)
VOS zu OS-VD und Ressourcenschema (VR)			
199	VOT:Fluginformation>	⇒	eVOT:Fluginformation>,VR:Fluginfo.,VR:Fluginfo.Liste
200	VOT:>Suchanfrage	⇒	eVOT:>Suchanfrage, VR:>Suchanfrage,VR:>Suchanfrageliste
201-230 VOTs		⇒	eVOTs, Vorgangsressourcen (Individual, Liste)
231	IOT:MCKE_Suchanfrage	⇒	R-IOT:MCKE_Suchanfrage
232	IOT:MCKS_Suchergebnis	⇒	R-IOT:MCKS_Suchergebnis
233-243 IOTs		⇒	R-IOTs
244	Interacts_with(IOT:MCKE_ Suchanfr.,VOT:> Suchanfrage)	⇒	Interacts_with (R-IOT:MCKE_Suchanfrage, eVOT: >Suchanfrage)
245	Interacts_with(VOT:> Such-anfrage , VOT:Flugstatus>)	⇒	Interacts_with (eVOT:>Suchanfrage, eVOT:Flug-status>)
246-284 Beziehungen		⇒	Beziehungen OS-VD
285	KOT:Auftragskopf, KOT:Auftragsposition	⇒	DokT:Auftrag: Flugwahl, Angebot, Auftragsdetails (Bestätigung, Sitzplatzwahl, Passagierdaten), Auftragszusammenfassung, Buchungsdetails (Buchung, Zahlungsdaten), Buchungsbestätigung, Reservierung, Reservierungsbestätigung [75]
286	KOT:Rechnungskopf, KOT:Rechnungsposition	⇒	DokT:Rechnung: Abrechnungsauftrag, Rechnung, Zahlung, Abrechnungsmeldung
287	KOT:Lieferung, KOT:eTicket	⇒	DokT:Lieferung, Lieferauftrag, eTicket, Lieferbest.
288	TOT:Flug, TOT:Sitzplatz	⇒	DokT:Flug(Sitzplatz)
289	TOT:Flugdaten (transient)	⇒	DokT:Flugdaten(Liste Flüge)
290	TOT:Suchanfrage (transient)	⇒	DokT:Suchanfrage
291	TOT:Flugstatus (transient)	⇒	DokT:Flugstatus
292	TOT:Suchergebnis (transient)	⇒	DokT:Suchergebnis

Tabelle A.5: Dokumentation von T2 der Fallstudie (konsolidiertes KOS und konsolidiertes VOS in die initialen Schemata des REST-Architekturmodells)

Nr	**V**	**B**	**N**
Keine Strukturänderungen am OS-ED, Detaillierung einzelner ERs			
293	ER:Auftragskopf, ER:Auftragskopfliste	→	ER:Auftragskopf, ER:Auftragskopfliste, ER:Auftragskopf-Status, ER:Stornierung, ER:Stornierungsliste
294	ER:Auftragsposition, ER:Auftragspositionsliste	→	ER:Auftragsposition, ER:Auftragspositionsliste (--- *keine Änderung* ---)
295	ER:Flug, ER:Flugliste	→	ER:Flug, ER:Flugliste, ER:Flugliste-Sucheingabe(Parameter)

Tabelle A.6: Dokumentation von K2.1 der Fallstudie (Überarbeitung OS-ED und RS-ER)

Nr	*OS-VD*	**V** (initial eVOT)	**B**	**N** (kons. eVOT)	*Überarbeitung (K2.2.1)*
296	Vertrieb	>Fluganfrage, Flugstatus>, >Flugdaten, >Suchergebnis	→	Flugsuche	Zusammenfassen (MEP: In-Out)
297		>Flugwahl,	→	Angebotsabwicklung	Zusammenfassen

[75] Die Basis bildet das dokumentenorientierte KOS nach K1 (vgl. KOTs des initialen KOS in Abbildung A.2)

	Vertrieb	Angebot>			(MEP: In-Out)
298		Auftragszusammenf.>, >Auftragsdetails	→	Auftragserstellung	Zusammenfassen (MEP: In-Out)
299		>Buchungsdetails	→	>Buchungsdetails	*(keine Änderung)*
300		Reservierung> >Reservierungsbestät.,	→	Reservierungs-beauftragung	Zusammenfassen (MEP: In-Only)
301		Buchungsbestätigung>	→	Buchungsbestätigung>	*(keine Änderung)*
302		Lieferauftrag>, >Lieferbestätigung	→	Versandbeauftragung	Zusammenfassen (MEP: In-Only)
303		Abrechnungsauftrag>, >Abrechnungsmeldung	→	Rechnungsbeauftragung	Zusammenfassen (MEP: In-Only)
304	Flugbetrieb	>Flugstatus, Flugdaten>	→	Flugauskunft	Zusammenfassen (MEP: In-Out)
305		>Reservierung, Reservierungsbestät.>	→	Reservierungs-abwicklung	Zusammenfassen (MEP: In-Out)
306	Versand	>Lieferauftrag, Lieferung eTickets>, Lieferbestätigung>	→	Versandabwicklung	Zusammenfassen der eVOTs des OS-VD Versand
307	Rechnungs-wesen	>Abrechnungsauftrag, Rechnung>, >Zahlung, Abrechnungsmeldung>	→	Rechnungsabwicklung	Zusammenfassen der eVOTs des OS-VD Rechn.w.
Nr	**V**		**B**	**N**	
	Anpassen und Überarbeiten von VRs und R-IOTs				
308	eVR:Angebot>, eVR:>Flugwahl, eVR:Angebot>Liste, eVR:>Flugwahlliste		→	eVR:>Angebotsabwicklung eVR:>Angebotsabwicklungsliste	
309	eVR:>Suchanfrage, eVR:Flugstatus>, eVR:>Flugdaten, eVR:Suchergebnis> eVR:(jeweilige Listenressourcen)		→	eVR:Flugsuche eVR:Flugsucheliste	
310-317 weitere Überarbeitung der eVRs von zusammengefassten eVOTs					
318	R-IOT:MCKE_Suchanfrage, R-IOT:MCKS_Suchergebnis		→	R-IOT:MCKE/S_Flugsuche	
319	R-IOT:MCKE_Angebot, R-IOT:MCKS_Flugwahl		→	R-IOT:MCKE/S_Angebotsabwicklung	
320	R-IOT:MCKE_Auftragsdetails, R-IOT:MCKS_Auftragszusammenfassung		→	R-IOT:MCKE/S_Auftragserstellung	

Tabelle A.7: Dokumentation von K2.2 der Fallstudie (Überarbeitung OS-VD und RS-VR)

Nr	**V**	**B**	**N**
Für alle persistent (p) markierten EOTs. EOT bleibt jeweils erhalten (rechts)			
321	EOT:Person	→	DOT:Person [EOT:Kunde]
322	EOT:Kunde	→	DOT:Kunde
323	EOT:Passagier	→	DOT:Passagier
324	Is_a-Beziehung (EOT:Person, EOT:Kunde)	→	Generalisierung(DOT:Person, DOT:Kunde)
325	EOT:Flug	→	DOT:Flug
326	EOT:Sitzplatz	→	DOT:Sitzplatz
327	Is_part_of-Beziehung (EOT:Flug,EOT:Sitzplatz)	→	Beziehung (DOT:Flug,DOT:Sitzplatz)
328-338: EOTs		→	DOTs
339-355: EOT-Beziehungen		→	DOT-Beziehungen

Tabelle A.8: Dokumentation von K2.4 (Spezifikation des Datenschemas)

B Ergänzungen zur SOM-R-Methodik

B.1 Metamodell des Strukturierten Entity-Relationship-Modells

Die Gestaltung der persistenten Datenhaltungsebene der SOM-R-Methodik wird in der Fallstudie am Beispiel des Relationenmodells demonstriert (z. B. Abschnitte 5.3.5 und 5.5.5.4). Zur konzeptuellen Modellierung des Datenschemas kommt das *Strukturierte Entity-Relationship-Modell* (SERM) zum Einsatz. Das SERM ([Sinz87], [Sinz88b], [FeSi13, S. 158 ff.]) entstand aus der Motivation, die Modellierung komplexer Datenschemata in umfangreichen Anwendungsentwicklungsprojekten zu unterstützen [FeSi13, S. 158]. Das SERM-Metamodell ist in Abbildung B.1 dargestellt.

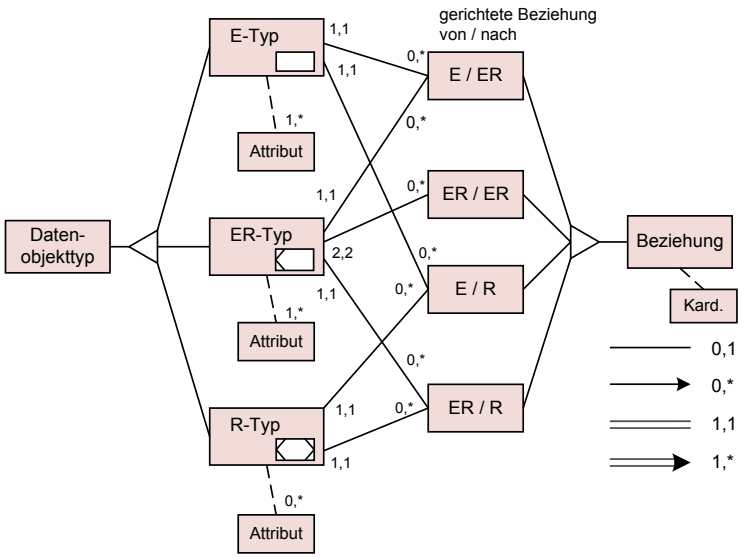


Abbildung B.1: Metamodell des Strukturierten Entity-Relationship-Modells (SERM) [FeSi13, S. 171]

Ein SERM-Schema besteht aus einer Menge von Datenobjekttypen und Beziehungen zwischen diesen. Ein *Datenobjekttyp* (DOT) ist entweder ein *Entity*- (E-), *Entity-Relationship*- (ER-) oder *Relationship-Typ* (R-Typ). E- und ER-Typen muss mindestens ein *Attribut* zugeordnet werden, während R-Typen Attribute enthalten können. Die Spezialisierungen der *Beziehungen* zwischen E-, ER- und R-Typen beschreiben zulässige Verknüpfungen zwischen Datenobjekttypen. Jede Beziehung weist eine *Kardinalität* in *(min,max)-Notation* auf. Die Standardkardinalitäten (0,1), (1,1), (0,*) und (1,*) werden jeweils durch eine eigene Kantenart symbolisiert [FeSi13, S. 171 f.]. Weiterführende Hinweise zu Darstellungs- und Modellierungsregeln im SERM finden sich in [FeSi13, S. 162 ff.].

B.2 Diskussion ausgewählter Entwicklungsschritte zur Modellbildung

B.2.1 Diskussion des Entwicklungsschrittes *Konsolidierung K1*

Die systematische Überbrückung der semantischen Lücke zwischen den Abstraktionsebenen *SOM-Geschäftsprozessmodell* und *softwaretechnisches REST-Architekturmodell* ist ein Ziel der Entwicklungsmethodik. Die Modellebene *fachliche AwS-Spezifikation* unterstützt in diesem Zusammenhang eine Überführung der methodischen Konzepte „betriebliches Objekt“ und „transaktionsorientierte Koordination“ des SOM-GPM in eine objektorientierte AwS-Spezifikation auf fachlicher, plattformunabhängiger AT-Ebene. Ausgewählte charakteristische Merkmale der ersten Konsolidierung werden im Folgenden noch einmal genauer untersucht.

- *Welche Informationen beschreibt das dokumentenorientierte KOS als „Zwischenergebnis“ der KOS-Konsolidierung?* Die Durchführung eines konkreten Geschäftsvorfalls wird in SOM als Folge von Transaktionen zwischen betrieblichen Objekten beschrieben. Der Baustein *betriebliche Transaktion* stellt im SOM-GPM zudem das Bindeglied zwischen den Sichten IAS und VES dar (Abbildung 4.5). Auf der fachlichen AwS-Ebene werden betriebliche Transaktionen als *transaktionsspezifische Objekttypen* des KOS abgebildet und sind Ausgangspunkt für *Interaktionsbeziehungen* und *Nachrichtendefinitionen* zwischen den VOTs unterschiedlicher VOS. Aus diesem Grund bilden TOTs die Grundlage für eine strukturelle Beschreibung der zwischen VOTs ausgetauschten Nachrichten (K1.2, Schritt 3). Diese Abhängigkeit wird im *dokumentenorientierten Stand des KOS* explizit erfasst und gekapselt.
- *Weshalb wird die Bestimmung der persistenten (p) und transienten (t) konzeptuellen Objekttypen als neuer Schritt in die Konsolidierung K1 aufgenommen?* Zusätzlich zu den Schritten der KOS-Konsolidierung in der SOM-Methodik wird die Bestimmung von Persistenzeigenschaften (p, t) konzeptueller Objekttypen auf fachlicher AT-Ebene vorgenommen. Die Definition eines eigenen Überarbeitungsschrittes betont die Relevanz von Transaktionen für die fachliche Spezifikation von Struktur und Verhalten des AwS in dieser Arbeit. Zudem wird die Beziehung zwischen den ausgetauschten Nachrichten der VOTs und der hierbei übertragenen Informationen (Dokumente) explizit beschrieben. Die Festlegung der Persistenzeigenschaft bereitet weiter die Spezifikation der Datenhaltungsschicht auf softwaretechnischer Ebene vor.
  Die Identifikation von transienten und persistenten Objekttypen lässt sich durch die Einbeziehung von Merkmalen der SOM-Geschäftsprozessebene unterstützen [WoBe13, S. 1234].
  - Betriebliche Objekte und Leistungsspezifikationen des SOM-GPM geben häufig Hinweise auf *Stammdaten*, die von Objekttypen persistent zu verwalten sind.
  - Den Ausführungszustand eines konkreten Geschäftsvorfalls erfassen *persistente Transaktionsdaten*, die durch eine Menge von Transaktionen zwischen betrieblichen Objekten (transaktionsorientierte Koordination und Durchführung der Leistungserstellung) beschrieben werden. Für eine Zusammenfassung von persistenten TOTs zu einem gemeinsamen TOT können die Unterscheidungsmerkmale „Art der Transaktion“ (AVD, SK) und beteiligte „betriebliche Objekte“ als Orientierungshilfe herangezogen werden. Beispielsweise geben die sequentiell ausgetauschten V-Transaktionen *Anfrage*, *Ange-*

bot, *Buchung* und *Bestätigung* zwischen zwei betrieblichen Objekten *Kunde* und *Vertrieb* Hinweise auf die Möglichkeit zur Bildung eines gemeinsamen TOT (z. B. als Objekttyp *Auftrag* mit den Status *Anfrage*, *Angebot*, *Buchung* und *Bestätigung*, vgl. auch Fallstudie, S. 121).

- *Transiente Transaktionsdaten* beschreiben einen nicht „persistierungswürdigen" Zustand im Geschäftsablauf.
  Als Beispiel seien übermittelte Kriterien einer Suchanfrage (V-Transaktion) genannt. Der Zustand der Interaktion wird in diesem Fall (durch den TOT) nicht dauerhaft gespeichert.

- *Welche Rolle spielt der Interface-Objekttyp bei der Bestimmung und Spezifikation von Kommunikationskanälen zur Umwelt?* Jeder IOT modelliert einen Kommunikationskanal für den Empfang oder das Senden von Nachrichten zu Objekten der Umwelt oder der Diskurswelt, sofern die Diskurswelttobjekte von verschiedenartigen Aufgabenträgern unterstützt werden. Zielsetzung der vorliegenden Arbeit ist die vollständige Unterstützung von Geschäftsprozessen durch ein AwS, das als RESTful SOA realisiert ist. Die RESTful SOA stellt damit den einzigen Aufgabenträger innerhalb der Diskurswelt dar. Die Zuordnung von IOTs ist folglich ausschließlich für eine Spezifikation von Kommunikationskanälen zur Umwelt notwendig.
  Ein sendender IOT formatiert und übermittelt für einen VOT Nachrichten an maschinelle (CCKS) oder personelle Aufgabenträger (MCKS). Entsprechend wird der Empfang von externen Nachrichten an einen VOT durch empfangende IOTs realisiert (CCKE oder MCKE) (vgl. [Mali97, S. 59 f.]). Demnach werden jedem VOT zur Kommunikation mit der Umwelt grundsätzlich zwei IOTs (S, E) zugeordnet. Aufgrund dieser starken Abhängigkeitsbeziehung zwischen IOT und VOT erfolgt die detaillierte Modellierung von Kommunikationsbeziehungen, ausgetauschten Nachrichten sowie IOTs im VOS. Das IOS ist als Projektion auf die IOTs des VOS beschreibbar. Sie wird im weiteren Verlauf jedoch nicht in einem eigenen Schema erfasst.

B.2.2 Diskussion des Entwicklungsschrittes *Transformation T2*

Die Überbrückung der Abstraktionslücke zwischen den Objektschemata der fachlichen AwS-Spezifikation und der initialen Spezifikation der RESTful SOA ist das Ziel der Gestaltung des Entwicklungsschrittes „Transformation T2". Die automatisierte Durchführung der Transformation setzt die Bestimmung einer konkreten Vorgehensweise voraus, zu der eine Reihe von Alternativen bestehen. Die Lösungsalternativen sowie die getroffenen Gestaltungsentscheidungen dieser Arbeit werden im Folgenden erläutert:

- *Warum werden Ressourcen und Dokumente der REST-Schnittstelle nicht aus den konsolidierten Schemata OS-ED und OS-VD abgeleitet?* Für eine Identifikation und Spezifikation von Ressourcen und Dokumenten der RESTful SOA lassen sich zwei grundlegende Vorgehensweisen identifizieren. Zunächst kann eine initiale Ableitung der REST-Schnittstelle auf Basis der Spezifikationen „konsolidiertes KOS" und „konsolidiertes VOS" erfolgen (fachliche Ebene) (vgl. [Wolf12], [WoBe13]). Alternativ bietet sich eine Transformation aus dem *konsolidierten OS-ED* und den *konsolidierten OS-VDs* an

(softwaretechnische Ebene).

Im Entwicklungsschritt „Transformation T2.2“ wird die initiale Ableitung und Spezifikation von Innen- und Außensicht der RESTful SOA aus der fachlichen Ebene vorgeschlagen. Als Grund hierfür ist anzuführen, dass im Rahmen der nachfolgenden Konsolidierung des REST-Architekturmodells (K2, Abschnitt 5.5.5) die vorliegenden Informationen aus Ressourcen- und Dokumentenschema in die weiteren Entwurfsentscheidungen zur Gestaltung der RESTful SOA einfließen können. Zudem würde im Fall, dass keine oder nur geringe strukturelle Anpassungen an den Objektschemata vorgenommen werden, eine Ableitung von Ressourcen und Dokumenten aus den konsolidierten Schemata OS-ED oder OS-VD keine wesentlichen Unterschiede zur initialen Transformation aufweisen (vgl. Schritt von K2.1 und K2.2 in den Abschnitten 5.5.5.1 bzw. 5.5.5.2).

Um den Nachteil einer verringerten Abstimmung zwischen Innen- und Außensicht zu vermeiden, ist beispielsweise eine Anwendung der Ableitungsregeln von T2.2 für jede strukturelle Anpassung im Zuge der Konsolidierung K2 denkbar. Eine abschließende Detaillierung der Spezifikation von Ressourcen und Dokumenten muss jedoch stets durch den Entwickler vorgenommen werden.

- *Warum werden das initiale Ressourcen- und initiale Dokumentenschema aus den Schemata der fachlichen Ebene abgeleitet?* Im Zuge der Transformation T2.2 werden das konsolidierte KOS und konsolidierte VOS in eine initiale Spezifikation von Ressourcen- und Dokumentenschema überführt. Alle Ableitungsregeln und Abbildungsbeziehungen, die für eine Überbrückung der dabei bestehenden Abstraktionslücke benötigt werden, sind durch die konzipierte Transformation gekapselt. Durch diesen Ansatz lassen sich alle Bestandteile der softwaretechnischen und fachlichen Ebene direkt zueinander in Beziehung setzen. Die initial abgeleiteten Dokumente und Ressourcen beschreiben jedoch die Außensicht der RESTful SOA für die Objekttypen von initialem OS-ED und initialem OS-VD. Aus diesem Grund ist auch ein Heranziehen dieser Schemata als Quelle von Transformation T2.2 denkbar, zumal die Objektschemata der fachlichen und softwaretechnischen Ebene in ihrer Modellstruktur übereinstimmen (1:1-Abbildung).

  Die Ableitung von Ressourcen und Dokumenten aus den initialen oder auch den konsolidierten Objektschemata des REST-Architekturmodells stellt somit eine mögliche Alternative zur vorgeschlagenen Vorgehensweise dar. Für jeden Ausführungszustand müsste in diesem Fall im Rahmen der Konsolidierung K2 ein Dokumenttyp erstellt werden. Insbesondere in Zusammenhang mit einer Entscheidung zur Ableitung von Dokumenten und Ressourcen aus dem konsolidierten OS-ED und konsolidierten OS-VD bietet sich die nähere Betrachtung eines alternativen Vorgehens an.

- *Warum sieht die Transformation T2 keine Ableitung von nicht-elementaren Vorgangsressourcen vor?* Jedes VOS spezifiziert das AwS für ein oder mehrere betriebliche Objekte als Menge fachlich zusammengehöriger VOTs, die untereinander in Beziehung stehen. Grundsätzlich ist die Interpretation jedes VOS als *nicht-elementarer Vorgangsdienst* (neVD) möglich, welcher die Ausführung „seiner“ elementaren Vorgangsdienste (eVD) orchestriert.

  Auf die Spezifikation einer standardisierten Ableitungsregel von neVD und ihrer Ressourcen wird in der SOM-R-Methodik verzichtet, da die Transformation einer Entwurfsentscheidung zur hierarchischen Koordination (Orchestrierung) aller eVDs innerhalb der

RESTful SOA gleichkommen würde. Die Wahl und Gestaltung der Koordination zwischen Vorgangsdiensten wird in der vorliegenden Arbeit jedoch als (nicht-automatisierbare) Entscheidung gesehen, die eine Abwägung von Vor- und Nachteilen bezüglich der Identifikation und Spezifikation von neVDs bzw. der nicht-hierarchischen Koordination von eVDs für das zu entwickelnde AwS erfordert (Abschnitt 5.5.5.2).

- *Wann ist eine Ableitung des initialen Dokumentenschemas aus dem dokumentorientierten KOS sinnvoll?* In Transformationsschritt T2.2 wird die Identifikation und Spezifikation von Dokumenttypen aus den Objekttypen des konsolidierten KOS vorgeschlagen. Eine Ableitung erfolgt dabei unter der Annahme, dass die Attributmengen der zusammengefassten TOTs (K1.1, Schritt 6) keine wesentlichen Strukturunterschiede aufweisen und somit eine sinnvolle Differenzierung ihres Zustandes erfolgen kann (z. B. über ein „Status-Attribut“ wie in der Fallstudie, vgl. S. 133).
  Alternativ zu diesem Vorgehen ist auch das Erfassen einzelner Ausführungszustände in einem eigenen Dokumenttyp möglich. Bei einem solchen Vorgehen würde für jeden KOT ein eigener Dokumenttyp definiert. Ausgangsbasis für die Durchführung der Transformation wären dann das *dokumentenorientierte KOS*. WOLF und BENKER schlagen in diesem Kontext eine *dokumentenorientierte Konsolidierung* des KOS vor [WoBe13].

- *Wie können die spezifizierten Dokumente und Ressourcen zur einheitlichen REST-Schnittstelle verknüpft werden?* Das Dokumentenschema beschreibt die Struktur der Repräsentationen von Ressourcen einer RESTful SOA in initialer Form. Die abgeleiteten Stammdaten- und Transaktionsdatendokumente definieren dabei Daten, welche die von EOTs bereitgestellten Strukturinformationen repräsentieren und mittels HTTP-Nachrichten über die einheitliche Schnittstelle der Objekttypen ausgetauscht werden. Da sowohl die Interaktionsbeziehungen im OS-VD, als auch die TOTs des OS-ED ihren Ursprung in den betrieblichen Transaktionen des SOM-Geschäftsprozessmodells haben, ist auf Basis dieser Beziehung eine Zuordnung von Dokumenttypen zu eingehenden (IN) oder ausgehenden (OUT) Nachrichten der Standard-(HTTP-)Operatoren von Ressourcen im initialen Ressourcenschema möglich.
  Eine Verknüpfung von Dokumenttypen mit den HTTP-Methoden des Ressourcenschemas erfolgt in der Konsolidierung K2. Die Beziehungen zwischen Nachrichten und Objekttypen sind auf der fachlichen Modellebene nicht explizit erfasst. Außerdem erfordert die Definition der Methodennachrichten normalerweise eine Konkretisierung ihrer Spezifikation, sowie der, im Rahmen eines Methodenaufrufs, ausgetauschten Informationen.

B.2.3 Diskussion des Entwicklungsschrittes *Konsolidierung K2*

Die Spezifikation der RESTful SOA auf softwaretechnischer Ebene ist Ziel der Konsolidierung des *REST-Architekturmodells*. Im Folgenden werden ausgewählte Merkmale des Entwicklungsschrittes noch einmal genauer erläutert.

- *Zusammenfassende Erläuterung des gewählten Vorgehens zur Durchführung von „Konsolidierung K2“*. Das Vorgehen zur Konsolidierung der softwaretechnischen Modellebene sieht die schrittweise Spezifikation der Teilsysteme Anwendungsfunktionalität, Kommunikation und Datenverwaltung der RESTful SOA vor (AKD-Strukturmodell, Abschnitt 2.2.4.3). Die

Innen- und Außensicht der Teilsysteme werden gemeinsam entwickelt, da sich die Entwurfsentscheidungen gegenseitig beeinflussen (vgl. K2.1, S. 131 und K2.2, S. 133).

a. Die Konsolidierung beginnt mit den initialen Schemata der *Anwendungsfunktionalität*. Auf dieser Basis können weitere Anforderungen an die Gestaltung der unterstützenden K- und D-Teilsysteme aufgedeckt werden. Das Vorgehen definiert zunächst die Überarbeitung und Anreicherung von OS-ED und Ressourcenschema (ER), die das REST-Architekturmodell primär aus *strukturorientierter Perspektive* betrachten. Aufbauend auf diesen Ergebnissen erfolgt die Konsolidierung von OS-VD und Ressourcenschema (VR) zur softwaretechnischen Spezifikation des *Systemverhaltens*. Die Konsolidierung der strukturorientierten Modelle (OS-ED und RS-ER) ist Voraussetzung für eine Überarbeitung der Vorgangsdienste, da diese ihr Lösungsverfahren als Navigation durch eine Menge von Entitätsdiensten realisieren. Die Spezifikation der Schemata umfasst die Überarbeitung der Modellstruktur und eine anschließende softwaretechnische Anreicherung der Bausteineigenschaften.

b. Die Konsolidierung des *Kommunikationsteils* der RESTful SOA sieht eine Abstimmung der Spezifikationen von IOTs mit den VOTs der OS-VDs vor.

c. Die Anforderungen an die Struktur der ausgetauschten Dokumente ergeben sich aus Nachrichtendefinitionen und Repräsentationen der spezifizierten Ressourcen. Sie fließen in die Überarbeitung des initialen Dokumentenschemas ein. Dieses erfasst damit alle Zustandsinformationen, die in den vorausgegangenen Entwicklungsschritten aufgedeckt wurden.

d. Die Ableitung und Spezifikation des konzeptuellen Datenschemas zur Beschreibung des *Datenverwaltungsteils* erfolgt auf Basis der konsolidierten Schemata des Anwendungs- und Kommunikationsteils des AwS sowie des Dokumentenschemas.

Aufgrund der Abhängigkeiten zwischen den einzelnen Modellen läuft die Entwicklung jedoch nicht streng sequentiell ab. Die konsistente Abstimmung der Schemata kann das Rückspringen auf vorherige Überarbeitungsschritte oder das wiederholte Durchlaufen eines oder mehrerer Konsolidierungsschritte erforderlich machen.

- *Welche Bedeutung liegt der einheitlichen REST-Schnittstelle von Vorgangsdiensten zugrunde*? Ein Vorgangsdienst der RESTful SOA kapselt das Lösungsverfahren zur Durchführung eines oder mehrerer Vorgänge in Form einer Workflowspezifikation. Jeder Vorgangsdienst stellt seine Funktionalität standardmäßig über eine Listenressource und Individualressource mit den HTTP-Methoden GET, POST bzw. GET, PUT und DELETE bereit. Jede Instanz der Individualressource entspricht somit einem konkreten Workflow (Prozessinstanz), dessen Ausführung über die einheitliche REST-Schnittstelle verwaltet wird. Während GET und PUT den Abruf bzw. die Aktualisierung des Ausführungszustands einer Workflowinstanz ermöglichen, kann ihre Durchführung mittels DELETE gestoppt bzw. angehalten werden. Ein GET auf die zugehörige Listenressource liefert alle Prozessinstanzen zu einer Workflowspezifikation zurück. Mit POST erfolgt die Erstellung einer neuen Prozessinstanz (z. B. API des WfMS *Activiti*, http://activiti.org/userguide/, Abruf am 11. Januar 2015).

Die **Konzeptionsentscheidungen von Konsolidierung K2.1** werden nachfolgend begründet sowie die Durchführung und Gestaltung einzelner Überarbeitungsschritte genauer erläutert:

- Aufgrund der gegenseitigen Abhängigkeit zwischen den Überarbeitungsschritten dienen die Schrittnummern in K2.1 (und auch K2.2) nur als logische Ordnungszahlen. Die Vorgabe einer festen Ausführungsreihenfolge ist an dieser Stelle nicht möglich.

- Schritt 1: Die explizite Darstellung von bisher nur transitiv beschriebenen Beziehungen unterstützt im weiteren Verlauf der Konsolidierung die Erstellung von Hyperlinks in Ressourcen-Repräsentationen und damit den Aufbau einer *Hypermedia* (K2.1, Schritte 5 und 6 sowie K2.2, Schritt 8). Es werden nur diejenigen Beziehungen hinzugefügt, die aus Sicht der Systementwicklung für den Aufbau der Hypermedia relevant sind.

- Schritt 2: Eine zielgerichtete Nutzung von Diensten der RESTful SOA setzt die Definition von URLs als Startpunkte der Anwendung voraus. Hierfür werden in der Methodik meist EOTs herangezogen, die aus objektspezifischen Objekttypen und damit aus betrieblichen Objekten des Geschäftsprozesses hervorgegangen sind. Die OOTs des KOS erlauben auch eine fachliche Begründung des Einstiegs in das System, da sie sich als VOS in der Verhaltenssicht wiederfinden.
  Im Beispiel der Fallstudie (Abbildung 5.15) wird das betriebliche Objekt (bzw. der abgeleitete EOT) *Vertrieb* als Host (*/vertrieb/*) für einen geeigneten Einstieg in die RESTful SOA interpretiert, da es sowohl im VOS die Anwendungsfunktionalität für Kunden-Bestellungen kapselt, also auch ein existenzunabhängiger Objekttyp im KOS ist. Die Bestimmung der Einstiegspunkte erfolgt in Abstimmung mit der Konsolidierung des OS-VD (K2.2.1).

- Schritt 4: Zur Bereitstellung der Funktionalität fachlicher Operationen eines Objekttypen sind häufig weitere Ressourcen zu definieren, falls diese nicht über die Standard-Methoden der initialen Ressource realisierbar sind.
  Fachliche Operationen können z. B. über die Spezifikation einer neuen Subressource abgebildet werden (z. B. */ressource/{id}/funktion*). Filterressourcen (URL-Schema: */ressource/_?attribut=parameter*) können zudem für die Selektion von Ressourceninstanzen eingesetzt werden, die eine bestimmte Eigenschaft oder einen bestimmten Zustand aufweisen (Abschnitt 3.3.2.1).

- Schritt 8: Zusammengesetzte Datentypen (Recordtyp) realisieren häufig Attribute in unterschiedlichen Objekttypen. Zur Vermeidung redundanter Definitionen ist die Kapselung des Datentyps als *Nicht-Objekttyp* möglich.
  In der Fallstudie (Abbildung 5.15) bietet sich z. B. die Erstellung des Recordtyps *Adresse* zur Verwendung in einer Vielzahl von Dokumenten und dem Datenschema an.

- Schritte 11-13: Auf Basis der überarbeiteten Eigenschaften von EOTs erfolgt eine Detaillierung der REST-Schnittstelle und Abstimmung von Innen- und Außensicht des AwS (Teilsystem der Entitätsdienste). Die Spezifikation des Ressourcenschemas spiegelt sich immer in den Objekttypeigenschaften wie Attributen und Methoden (Parametern) wider (Überschneidung mit den Schritten 8 und 9).

Die **Konzeptionsentscheidungen der Konsolidierung K2.2.1** wird nun genauer begründet und die Durchführung sowie Gestaltung einzelner Überarbeitungsschritte tiefergehend erläutert:

- *Schritt 1*: Als Ergebnis der Transformation T2 realisiert jeder eVOT zunächst einen Vorgangsdienst, dessen Ressourcen ausgehend von den VOTs der fachlichen Modellebene in initialer Form abgeleitet werden. Auf Basis der eingehenden und ausgehenden Nachrichten zwischen eVOTs wird die initiale Spezifikation des OS-VD überprüft und bei Bedarf überarbeitet.
  Beispielsweise bietet sich die Zusammenfassung von eVOTs an, um die Realisierung einer fachlich abgeschlossenen Funktion der Geschäftslogik über einem gemeinsamen Dienst bereitzustellen. So können *>Flugstatus* und *Flugdaten>* des OS-VD *Flugbetrieb* zum elementaren Vorgangsobjekttypen *Flugprüfung* zusammengefasst werden (vgl. Fallstudie, Abbildung 5.16). Die Eigenschaften der hierbei zusammengefassten eVOTs werden übernommen, angepasst und über neu definierte Ressourcen bereitgestellt.
- *Schritt 2*: Bezüglich der Koordination von Vorgangsdiensten ist zunächst eine grundsätzliche Entscheidung für oder gegen eine zentrale Vorgangssteuerung zu treffen. Im Falle der Entscheidung für den Einsatz einer hierarchischen Dienstkoordination sind die nicht-elementaren VDs auf Basis des OS-VD zu identifizieren und die Orchestrierung der elementaren VDs zu spezifizieren. Die eVOTs und ihre Beziehungen im initialen OS-VD beschreiben logische Verknüpfungen zwischen Diensten im Sinne einer Choreographie und können deshalb übernommen werden.
- *Schritt 12*: Die Spezifikation der Lösungsverfahren von elementaren und nicht-elementaren VDs erfolgt aus der Innensicht unter Verwendung einer ausführbaren Workflowsprache. In der vorliegenden Arbeit wird hierfür die BPMN genutzt, deren Elemente um REST-spezifische Eigenschaften angereichert werden (Abschnitt 5.3.6).

B.3 Modellbasierte Spezifikation von RESTful Services mit Ruby on Rails

Das Web-Application-Framework *Ruby on Rails*[76] (kurz Rails oder auch RoR) wird im Folgenden als alternative Technologie für die Ausgestaltung und Durchführung der Schritte T3 und K3 zur Erstellung des Implementierungsmodells eingeführt. Die Wahl von Rails als zweite Programmierplattform erfolgt in dieser Arbeit, neben der bereits angeführten Demonstration der flexiblen Anpassbarkeit der SOM-R-Methodik und der Plattformunabhängigkeit des REST-Architekturmodells (Abschnitt 5.5.8), aus dem Grund, dass sich Rails-Anwendungen besonders für die modellgetriebene Entwicklung von RESTful Services anbieten. So veröffentlichen Rails-AwS ihre Funktionalität standardmäßig als Ressourcen mit einheitlicher REST-Schnittstelle (z. B. [RTH11, S. xviii f.]). Ein zentraler Bestandteil des Frameworks ist zudem ein Generator, der es ermöglicht, die Grundstruktur von RESTful Services automatisiert zu erzeugen (*Scaffolding*) [RTH11, S. 30].

[76] Aktuell liegt Rails in der Version 4.2 vor (Stand: 19. Dezember 2014).
Offizielle Webseite: http://rubyonrails.org/ (Abruf am 19. Dezember 2014).

Das Ruby on Rails-Framework wurde ursprünglich im Jahre 2004 von David Heinemeier Hansson entwickelt und ist, wie der Name angedeutet, in der Programmiersprache Ruby[77] geschrieben. Rails selbst liegt in Form eines Ruby-Softwarepaketes (sogenanntes Gem[78]) vor. Zur Realisierung seiner Funktionalität nutzt Rails wiederum weitere Gems, die über den Paketmanager *RubyGem* verwaltet werden ([Carl07, S. 152 ff.], [BiKa12, S. 2]).

Die Entwicklung von Rails-Anwendungen basiert auf den Grundprinzipien *Konvention über Konfiguration* sowie *DRY* (Don't Repeat Yourself). Die erstellten Anwendungen werden streng nach dem (Model-View-Controller-) *MVC-Pattern*[79] strukturiert [Carl07, S. 154 ff.].

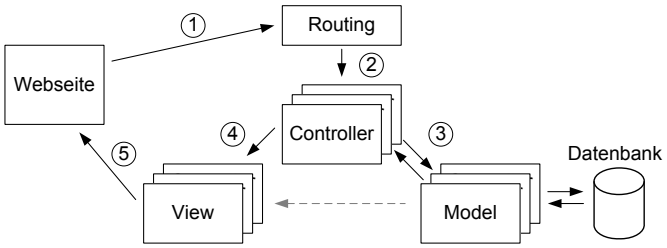


Abbildung B.2: MVC-Architektur in Rails (in Anlehnung an [RTH11, S. 31])

Weitere zentrale Hilfsmittel in Rails sind das bereits genannte *Scaffolding* zur automatisierten Erzeugung von Anwendungsgerüsten sowie ein *objektrelationaler Mapper*, der die Datenbankverwaltungslogik zu einem hohen Grad auf Anwendungsebene kapselt. Das grundlegende Zusammenspiel der MVC-Bestandteile in Rails zeigt Abbildung B.2.

Die HTTP-Anfragen des Clients (1), die i. d. R. über eine Webseite gesendet werden, leitet die *Routing*-Komponente an die entsprechenden Methoden (Actions) des *Controllers* weiter (2). Der aufgerufene Controller interagiert im Falle des Zugriffs und/oder Manipulation von Ressourcenzuständen mit einem Model. Das *Model* kapselt die Geschäftslogik zur Verwaltung persistierter Daten und greift auf die *Datenbank* mittels O/R-Mapper zu. Das Ergebnis der Anfrageverarbeitung reicht der Controller an den *View* weiter (4), der die Antwortdaten rendert und dem Client übermittelt (5) [RTH11, S. 31 f.].
Für detailliertere Informationen zu den Bestandteilen sowie der Funktionsweise des Rails-Pakets sei auf die weiterführende Literatur verwiesen (z. B. [RTH11], [BiKa12], [Wint13]).

Überführung des REST-Architekturmodells in eine Rails-Implementierung

Die Ableitungsschritte zur Transformation der Bausteine des REST-Architekturmodells in die Komponenten einer Rails-Anwendung fasst Abbildung B.3 zusammen.

---

[77] Die objektorientierte Programmiersprache Ruby wurde von Yukihiro Matsumoto entwickelt und 1995 veröffentlicht [FlMa08]. Aktuell liegt Ruby in der Version 2.2.1 vor (Stand: 2. März 2015).
Offizielle Webseite: https://www.ruby-lang.org/de/ (Abruf am 2. Oktober 2014).

[78] Damit stellt Rails nur eines von vielen Softwarepaketen dar, die in Ruby geschrieben sind. Die verfügbaren RubyGems finden sich unter https://rubygems.org/ (Abruf am 2. Oktober 2014).

[79] Das Model-View-Controller-Muster [Reen79] beschreibt die Strukturierung von Softwaresystemen in Model (Datenmanagementschicht), View (Präsentationsschicht) und Controller (Steuerungsschicht). Die Realisierung der anwendungsspezifischen Geschäftslogik einer Anwendung erfolgt, abhängig vom zugrunde gelegten Verständnis, durch Model und/oder Controller.

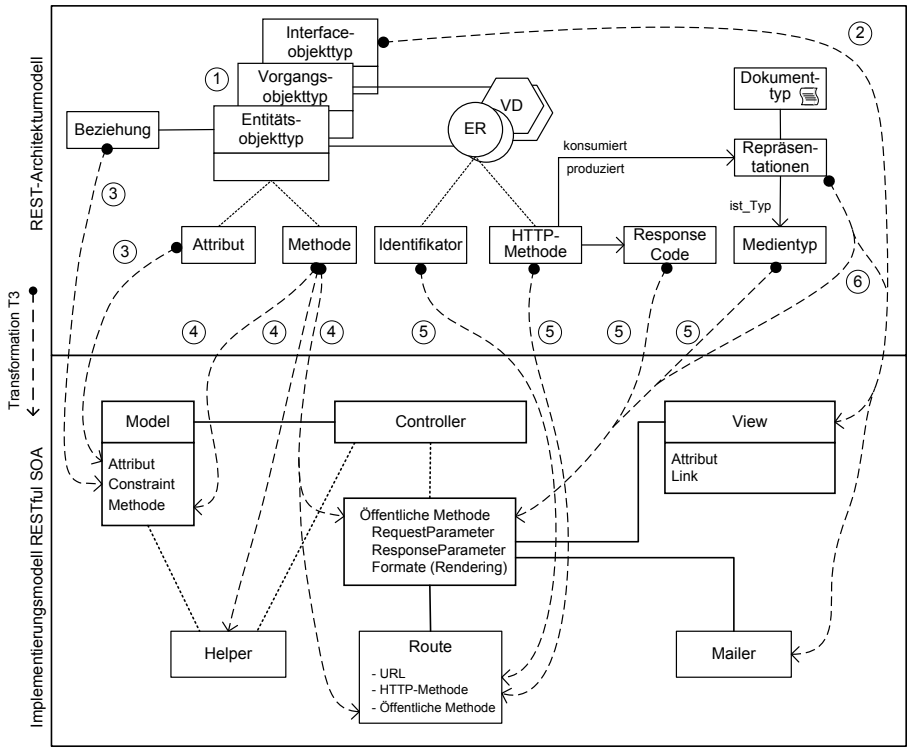


Abbildung B.3: Beziehungen zwischen den Metamodellen von REST-Architekturmodell und Rails-Implementierungsmodell

Die Transformation des REST-Architekturmodells in das Rails-Implementierungsmodell (T3) umfasst folgende Schritte:

1. Vorgangs- und Entitätsobjekttypen werden in Model- und Controller-Klassen sowie korrespondierende Views transformiert (nicht explizit darstellt).
2. Interfaceobjekttypen definieren Anforderungen zur Kommunikation mit externen Aufgabenträgern. Sie sind Basis für die Gestaltung der Views von Vorgangsobjekttypen oder die Erstellung alternativer Schnittstellenkomponenten (z. B. Mailer).
3. Die Struktur der Datenhaltungsebene wird in den Model-Klassen spezifiziert. Beziehungen zwischen Datenobjekttypen werden in „Table Constraints“ überführt und definieren die Abhängigkeiten im Datenbankschema der Anwendung. Die Attribute von EOTs, in Abstimmung mit Attributen korrespondierender DOTs, beschreiben die Attribute der Model-Klassen und damit auch der DB-Tabellen.
4. Private Methoden werden in Methoden des Models, Helpermethoden und in öffentliche Methoden in Controller-Actions überführt. Die Zuordnung von öffentlichen Methoden und HTTP-Methoden spezifiziert eine Route.
5. Die Eigenschaften der Ressourcenschnittstelle (Identifikator, HTTP-Methode, Medientypen etc.) beschreiben somit Routes und Controller-Actions der Anwendung.
6. Ressourcen-Repräsentationen (Dokumente) bestimmen den strukturellen Aufbau der Request- und Response-Parameter sowie der Views.

Der Einsatz des Framework Rails unterstützt die modellbasierte Entwicklung durch die Kapselung einer Vielzahl von Implementierungsaufgaben. Hierzu ist die Einhaltung der eingangs

genannten Grundprinzipien erforderlich. Für die Spezifikation der RESTful SOA bietet sich der Einsatz von Scaffolding zur automatisierten Erzeugung der Grundstruktur von RESTful Services an. Anschließend kann eine Anpassung der öffentlichen Schnittstelle und Realisierung der Lösungsverfahren durchgeführt werden. Bei der Überführung der BPMN-Workflows in eine plattformspezifische Implementierung gelten die Ausführungen des Abschnittes 5.5.8.1.

Eine initiale Version des Rails-Metamodells wurde in einer Abschlussarbeit vorgeschlagen (vgl. Abbildung B.4) [Roth12]. Sie dient als Basis für die Spezifikation der Rails-Zielarchitektur.

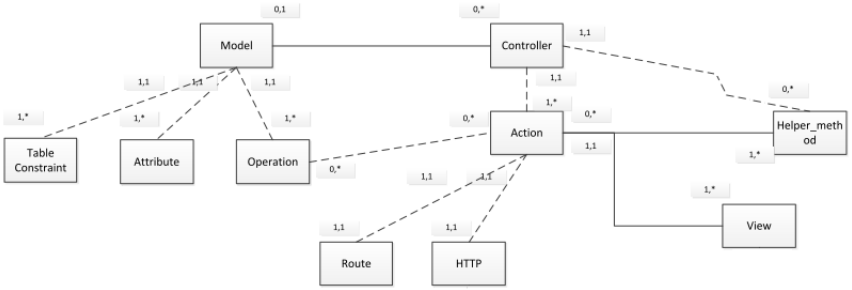


Abbildung B.4: Rails-Metamodell (nach [Roth12, S. 32])

Für den Einsatz in der SOM-R-Methodik werden Aufbau und Bestandteile des Metamodells überarbeitet und an die Begrifflichkeiten der SOM-R-Methodik angepasst.

SERVERKONFIGURATION UND DATENHALTUNG

Die Konfiguration von Web-, Anwendungs- und Datenbankserver wird im Rahmen der Systementwicklung größtenteils in das Rails-Framework ausgelagert. Als Web- und Applikationsserver werden standardmäßig *WEBrick* oder auch *Mongrel*[80] als Teil des Ruby-Pakets für Entwicklung und Test von Anwendungen eingesetzt. Durch den Einsatz des integrierten O/R-Mappers sind DB-Konfiguration und -Verwaltung sowie der Zugriff auf das Datenschema stark vereinfacht.

[80] Ruby-Gems Webseiten: https://rubygems.org/gems/webrick (WEBrick) und https://rubygems.org/gems/mongrel (Mongrel) (Abruf am 2. Oktober 2014)

Literaturverzeichnis

[ABB+09] Arsanjani, A.; Booch, G.; Boubez, T.; Brown, P. C.; Chappell, D.; deVadoss, J.; Erl, T.; Josuttis, N.; Krafzig, D.; Little, M.; Loesgen, B.; Manes, A. T.; McKendrick, J.; Ross-Talbot, S.; Tilkov, S.; Utschig-Utschig, C.; Wilhelmsen, H. (2009): SOA-Manifest. URL: http://www.soa-manifesto.org (Abruf: 17.04.2014).

[ABW11] Aier, S.; Bucher, T.; Winter, R. (2011): Kritische Erfolgsfaktoren für die Gestaltung serviceorientierter Informationssysteme. In: Wirtschaftsinformatik 53 (2), S. 75–87.

[Adam90] Adam, D. (1990): Integration und Flexibilität – Eine Herausforderung für die allgemeine Betriebswirtschaftslehre, 51. Wissenschaftliche Jahrestagung des Verbandes der Hochschullehrer für Betriebswirtschaftslehre [sic] e.V. 1989 in Münster. Gabler, Wiesbaden.

[AGA+08] Arsanjani, A.; Ghosh, S.; Allam, A.; Abollah, T.; Ganapathy, S.; Holley, K. (2008): SOMA: A method for developing service-oriented solutions. In: IBM Systems Journal 47 (3), S. 377–396.

[ALC08] Angyal, L.; Lengyel, L.; Charaf, H. (2008): A Synchronizing Technique for Syntactic Model-Code Round-Trip Engineering. In: Proceedings of the 15th Annual IEEE International Conference on Engineering of Computer Based Systems (SEDE-2006), Los Angeles, California, S. 463–472.

[Alge07] Algermissen, J. (2007): Serviceorientierung mit REST. In: Starke, G.; Tilkov, S. (Hrsg.): SOA-Expertenwissen – Methoden, Konzepte und Praxis serviceorientierter Architekturen. dpunkt, Heidelberg, S. 777–804.

[Alla10] Allamaraju, S. (2010): RESTful Web services cookbook – Solutions for Improving Scalability and Simplicity. O'Reilly, Sebastopol, Calif.

[Allw98] Allweyer, T. (1998): Adaptive Geschäftsprozesse – Rahmenkonzept und Informationssysteme. Gabler, Wiesbaden.

[AlPo03] Alanen, M.; Porres, I. (2003): Difference and union of models. TUCS technical report, Turku Centre for Computer Science, Turku.

[AlSc95] Allweyer, T.; Scheer, A.-W. (1995): Modellierung und Gestaltung adaptiver Geschäftsprozesse. In: Veröffentlichungen des Instituts für Wirtschaftsinformatik (Heft 115).

[Altr84] Altrogge, G. (1984): Flexibilität in der Produktion. In: Kern, W. (Hrsg.): Handwörterbuch der Produktionswirtschaft. Schäffer-Poeschel, Stuttgart, S. 604–618.

[AlWi10a] Alarcón, R.; Wilde, E. (2010a): Linking Data from RESTful Services. In: Bizer, C.; Heath, T.; Berners-Lee, T.; Hausenblas, M. (Hrsg.): Proceedings of the Linked Data on the Web Workshop (LDOW2010), April 27, 2010, Raleigh, North Carolina, Paper 10.

[AlWi10b] Alarcón, R.; Wilde, E. (2010b): RESTler: Crawling RESTful Services. In: Rappa, M.; Jones, P.; Freire, J.; Chakrabarti, S. (Hrsg.): Proceedings of the 19th International Conference on World Wide Web (WWW'10), Raleigh, North Carolina. ACM, New York, NY, S. 1051–1052.

[AnCz06] Antkiewicz, M.; Czarnecki, K. (2006): Framework-Specific Modeling Languages with Round-Trip Engineering. In: Hutchison, D.; Kanade, T.; Kittler, J.; Kleinberg, J.

M.; Mattern, F.; Mitchell, J. C.; Naor, M.; Nierstrasz, O.; Pandu Rangan, C.; Steffen, B.; Sudan, M.; Terzopoulos, D.; Tygar, D.; Vardi, M. Y.; Weikum, G.; Whittle, J.; Harel, D.; Reggio, G. (Hrsg.): Model Driven Engineering Languages and Systems. Springer, Berlin, Heidelberg, S. 692–706.

[ANR+06] Aizenbud-Reshef, N.; Nolan, B. T.; Rubin, J.; Shaham-Gafni, Y. (2006): Model traceability. In: IBM Systems Journal 45 (3), S. 515–526.

[Appl97] Appleton, B. (1997): Patterns and Software: Essential Concepts and Terminology. In: Object Magazine Online 3 (5), S. 20–25.

[Arms74] Armstrong, W. W. (1974): Dependency Structures of Data Base Relationships. In: Proceedings of the IFIP Congress, S. 580–583.

[Arsa04] Arsanjani, A. (2004): Service-oriented modeling and architecture – How to identify, specify, and realize services for your SOA.
URL: http://www.ibm.com/developerworks/library/ws-soa-design1/
(Abruf: 06.01.2013).

[AWB11] Alarcón, R.; Wilde, E.; Bellido, J. (2011): Hypermedia-driven RESTful service composition. In: Maximilien, E. M. (Hrsg.): Service-Oriented Computing. International Conference on Service-Oriented Computing (ICSOC 2010), San Francisco, CA, USA, December 7-10, 2010, Revised Selected Papers. Springer, Berlin, S. 111–120.

[AZE+07] Arsanjani, A.; Zhang, L.-J.; Ellis, M.; Allam, A.; Channabasavaiah, K. (2007): Design an SOA solution using a reference architecture. URL: http://www.ibm.com/developerworks/library/ar-archtemp/ (Abruf: 05.05.2014).

[BaGü09] Bauer, A.; Günzel, H. (2009): Data-Warehouse-Systeme – Architektur, Entwicklung, Anwendung. 3. Aufl., dpunkt, Heidelberg.

[Balz09] Balzert, H. (2009): Lehrbuch der Softwaretechnik – Basiskonzepte und Requirements Engineering. 3. Aufl., Spektrum Akademischer Verlag, Heidelberg.

[Balz11] Balzert, H. (2011): Lehrbuch der Softwaretechnik – Entwurf, Implementierung, Installation und Betrieb. 3. Aufl., Spektrum Akademischer Verlag, Heidelberg.

[BBC+04] Baïna, K.; Benatallah, B.; Casati, F.; Toumani, F. (2004): Model-Driven Web Service Development. In: Kanade, T.; Kittler, J.; Kleinberg, J. M.; Mattern, F.; Mitchell, J. C.; Naor, M.; Nierstrasz, O.; Pandu Rangan, C.; Steffen, B.; Sudan, M.; Terzopoulos, D.; Tygar, D.; Vardi, M. Y.; Weikum, G.; Persson, A.; Stirna, J. (Hrsg.): Advanced Information Systems Engineering. Springer, Berlin, Heidelberg, S. 290–306.

[BBF+11] Bartmann, D.; Bodendorf, F.; Ferstl, O. K.; Sinz, E. J. (2011): Merkmale, Systemarchitekturen und Management hochflexibler Geschäftsprozesse. In: Sinz, E. J.; Bartmann, D.; Bodendorf, F.; Ferstl, O. K. (Hrsg.): Dienstorientierte IT-Systeme für hochflexible Geschäftsprozesse. University of Bamberg Press, Bamberg, S. 1–13.

[BBW+05] Bieberstein, N.; Bose, S.; Walker, L.; Lynch, A. (2005): Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. In: IBM Systems Journal 44 (4), S. 691–708.

[BCK03] Bass, L.; Clements, P.; Kazman, R. (2003): Software architecture in practice. 2. Aufl., Addison-Wesley, New York.

[BCW12] Brambilla, M.; Cabot, J.; Wimmer, M. (2012): Model-driven software engineering in practice. Morgan & Claypool, San Rafael, Calif.

[Behr85] Behrbohm, P. (1985): Flexibilität in der industriellen Produktion – Grundüberlegungen zur Systematisierung und Gestaltung der produktionswirtschaftlichen Flexibilität. P. Lang, Frankfurt am Main, New York.

[BeJa09] Bernhard, R.; Jahn, B. U. (2009): Modellgetriebene Entwicklung von service-orientierten Architekturen. In: Hansen, H. R.; Karagiannis, D.; Fill, H.-G. (Hrsg.): Business services: Konzepte, Technologien, Anwendungen. 9. Internationale Tagung Wirtschaftsinformatik, Wien, 25.–27. Februar 2009, Österreich. Computer-Gesellschaft, Wien, S. 89–98.

[BeKa11] Becker, J.; Kahn, D. (2011): The Process in Focus. In: Becker, J.; Kugeler, M.; Rosemann, M. (Hrsg.): Process management. A guide for the design of business processes. 2. Aufl., Springer, Berlin [u.a.], S. 3–13.

[BePf06] Becker, J.; Pfeiffer, D. (2006): Konzeptionelle Modellierung - ein wissenschafts-theoretischer Forschungsleitfaden. In: Lehner, F.; Nösekabel, H.; Kleinschmidt, P. (Hrsg.): Tagungsband der Multikonferenz Wirtschaftsinformatik 2006. Gito, Berlin, S. 3–19.

[BER+10] Bhatt, G.; Emdad, A.; Roberts, N.; Grover, V. (2010): Building and leveraging information in dynamic environments: The role of IT infrastructure flexibility as enabler of organizational responsiveness and competitive advantage. In: Information & Management 47 (7-8), S. 341–349.

[BeRa00] Bennett, K. H.; Rajlich, V. T. (2000): Software maintenance and evolution: Proceedings of the Conference on The Future of Software (ICSE'00), Limerick, Ireland, S. 73–87.

[BFL+14] Berjon, R.; Faulkner, S.; Leithead, T.; Doyle Navara, E.; O'Connor, E.; Pfeiffer, S.; Hickson, I. (2014): HTML5 – A vocabulary and associated APIs for HTML and XHTML. W3C Candidate Recommendation 04 February 2014.
URL: http://www.w3.org/TR/html5/ (Abruf: 24.04.2014).

[BFM05] Berners-Lee, T.; Fielding, R. T.; Masinter, L. (2005): Uniform Resource Identifier (URI): Generic Syntax.
URL: http://tools.ietf.org/html/rfc3986/ (Abruf: 24.04.2014).

[BGM+10] Birbeck, M.; Gylling, M.; McCarron, S.; Pemberton, S. (2010): XHTML™ 2.0 – W3C Working Group Note 16 December 2010. URL: http://www.w3.org/TR/2010/_NOTE-xhtml2-20101216/ (Abruf: 07.05.2014).

[BHL+09] Bray, T.; Hollander, D.; Layman, A.; Tobin, R.; Thompson, H. (2009): Namespaces in XML 1.0 (Third Edition) – W3C Recommendation 8 December 2009. URL: http://www.w3.org/TR/2009/REC-xml-names-20091208/ (Abruf: 25.04.2014).

[BHM04] Booth, D.; Haas, H.; McCabe, F. (2004): Web Services Architecture.
URL: http://www.w3.org/TR/ws-arch/ (Abruf: 11.01.2014).

[BiKa12] Bigg, R.; Katz, Y. (2012): Rails 3 in action. Manning, Shelter Island, NY.

[BKO08] Birkmeier, D.; Klöckner, S.; Overhage, S. (2008): Zur systematischen Identifikation von Services: Kriterien, aktuelle Ansätze und Klassifikation. In: Loos, P.; Nüttgens, M.; Turowski, K.; Werth, D. (Hrsg.): Modellierung betrieblicher Informations-systeme (MobIS 2008). Modellierung zwischen SOA und Compliance Management, 27.-28. November 2008, Saarbrücken. Lecture Notes in Informatics (LNI) P-141. Köllen, Bonn, S. 255-272.

[BlSc06] Bloomberg, J.; Schmelzer, R. (2006): Service orient or be doomed! – How service orientation will change your business. Wiley, Hoboken, N.J.

[BoSi13] Bork, D.; Sinz, E. J. (2013): Bridging the Gap From a Multi-View Modelling Method to the Design of a Multi-View Modelling Tool. In: Enterprise Modelling and Information Systems Architectures 8 (2), S. 25-41.

[BPS+08] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; Yergeau, F. (2008): Extensible Markup Language (XML) 1.0 (Fifth Edition) – W3C Recommendation 26 November 2008. URL: http://www.w3.org/TR/2008/REC-xml-20081126/ (Abruf: 22.04.2014).

[BrPi08] Brun, C.; Pierantonio, A. (2008): Model Differences in the Eclipse Modelling Framework. In: UPGRADE IX (2), S. 29–34.

[BSL+08] Berenguer, I.; Shijun, C.; Liang, L.; Jing, L.; Wang, N. (2008): E-commerce at Yunnan Lucky Air. In: MITSloan Management Case Studies (08-076).

[BuMa08] Buzacott, J. A.; Mandelbaum, M. (2008): Flexibility in manufacturing and services: achievements, insights and challenges. In: Flexible Services and Manufacturing Journal 20 (1-2), S. 13–58.

[Burk13] Burke, B. (2013): RESTful Java with JAX-RS 2.0. 2. Aufl., O'Reilly, Sebastopol, CA.

[Burm02] Burmann, C. (2002): Strategische Flexibilität und Strategiewechsel als Determinanten des Unternehmenswertes. Deutscher Universitätsverlag, Wiesbaden.

[Busc00] Buschmann, F. (2000): Pattern-orientierte Softwarearchitektur – Ein Pattern-System. 1. Aufl. (korrigierter Nachdruck), Addison-Wesley, München.

[BWB11] Becker, A.; Widjaja, T.; Buxmann, P. (2011): Nutzenpotenziale und Herausforderungen des Einsatzes von Serviceorientierten Architekturen. In: Wirtschaftsinformatik 53 (4), S. 187–199.

[ByTu00] Byrd, T. A.; Turner, D. E. (2000): Measuring the Flexibility of Information Technology Infrastructure – Exploratory Analysis of a Construct. In: Journal of Management Information Systems 17 (1), S. 167–208.

[Carl06] Carl, D. (2006): Praxiswissen Ajax – Interaktive Web-Anwendungen mit Ajax; mit Einführungen in die Grundlagentechniken JavaScript DOM HTML und XML; praxisnahe Programmbeispiele im Buch. O'Reilly, Bejing.

[Carl07] Carl, D. (2007): Praxiswissen Ruby on Rails. O'Reilly, Beijing.

[CCY02] Cleland-Huang, J.; Chang, C.; Yujia Ge (2002): Supporting event based traceability through high-level recognition of change events: Proceedings of the 26th Annual International Computer Software and Applications Conference, August 26-29, 2002, Oxford, UK, S. 595–600.

[CDK+07] Curbera, F.; Duftler, M.; Khalaf, R.; Lovell, D. (2007): Bite: Workflow Composition for the Web. In: Krämer, B. J.; Lin, K.-J.; Narasimhan, P. (Hrsg.): Service-Oriented Computing – ICSOC 2007. Springer, Berlin, Heidelberg, S. 94–106.

[CDP07] Cicchetti, A.; Di Ruscio, D.; Pierantonio, A. (2007): A Metamodel Independent Approach to Difference Representation. In: Journal of Object Technology 6 (9), S. 165–185.

[Chen76] Chen P.P.-S (1976): The Entity-Relationship Model - Toward a Unified View of Data. In: ACM Transactions on Database Systems 1 (1), S. 9–36.

[CHL+07] Chinnici, R.; Haas, H.; Lewis, A. A.; Moreau, J.-J.; Orchard, D.; Weerawarana, S. (2007): Web Services Description Language (WSDL) Version 2.0 – Part 2 Adjuncts: W3C Recommendation 26 June 2007. URL: http://www.w3.org/TR/wsdl20-adjuncts/ (Abruf: 12.06.2014).

[CMR+07] Chinnici, R.; Moreau, J.-J.; Ryman, A.; Weerawarana, S. (2007): Web Services Description Language (WSDL) Version 2.0 – Part 1 Core Language: W3C Recommendation. URL: http://www.w3.org/TR/wsdl20/ (Abruf: 17.04.2014).

[CMS06] Castro, V. de; Marcos, E.; Sanz, M. L. (2006): A model driven method for service composition modelling: a case study. In: International Journal of Web Engineering and Technology (IJWET) 2 (4), S. 335.

[CMV10] Castro, V. de; Marcos, E.; Vara, J. M. (2010): Applying CIM-to-PIM model transformations for the service-oriented development of information systems. In: Information and Software Technology 53 (1), S. 87–105.

[CMW09] Castro de, V.; Marcos, E.; Wieringa, R. (2009): Towards a Service-Oriented MDA-Based Approach to the Alignment of Business Processes with IT Systems – Form the Business Model to a Web Service Composition Model. In: International Journal of Cooperative Information Systems 18 (2), S. 225–260.

[Codd70] Codd, E. F. (1970): A relational model of data for large shared data banks. In: Communications of the ACM 13 (6), S. 377–387.

[CRB03] Chung, S. H.; Rainer, R. Kelly Jr; Bruce, L. R. (2003): The Impact of Information Technology Infrastructure Flexibility on Strategic Alignment and Applications Implementation. In: Communications of the Association for Information Systems 11, S. 1–27.

[Croc06] Crockford, D. (2006): The application/json Media Type for JavaScript Object Notation (JSON) – RFC 4627. URL: http://tools.ietf.org/html/rfc4627/ (Abruf: 28.04.2014).

[CzEi05] Czarnecki, K.; Eisenecker, U. W. (2005): Generative programming – Methods, tools, and applications. 6. Aufl., Addison-Wesley, Boston.

[Davi12] Davis, C. (2012): What if the Web Were not RESTful? In: Alarcon, R.; Pautasso, C.; Wilde, E. (Hrsg.): Proceedings of the Third International Workshop on RESTful Design (WS-REST 2012), April 17, 2012, Lyon, France, S. 3–10.

[DBW+10] Dorn, C.; Burkhart, T.; Werth, D-; Dustdar, S. (2010): Self-adjusting Recommendations for People-Driven Ad-Hoc Processes. In: Hull, R.; Mendling, J.; Tai, S. (Hrsg.): Proceedings of the 8th International Conference on Business process management (BPM 2010), September 13–16, 2010, Hoboken, NJ, USA. Springer, Berlin, S. 327-342.

[Dick02] Dick, J. (2002): Rich Traceability. In: Spanoudakis, G.; Zisman, A.; Perez-Minana, E. (Hrsg.): Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'02), 2002, Edinburgh, Scotland.

[DLO+09] Decker, G.; Lüders, A.; Overdick, H.; Schlichting, K.; Weske, M. (2009): RESTful Petri Net Execution. In: Roberto Bruni; Karsten Wolf (Hrsg.): Proceedings of the 5th International Workshop on Web Services and Formal Methods (WS-FM 2008), September 4-5, 2008, Milan, Italy. Springer, S. 73–87.

[DRG+09] Delgado, A.; Ruiz, F.; Guzmán, G.-R.; Piattini, M. (2009): Towards a Service-oriented and Model-driven Framework with Business Processes as First-class Citizens. In: Witold Abramowicz; Leszek A. Maciaszek; Ryszard Kowalczyk; Andreas Speck (Hrsg.): Business Process, Services Computing and Intelligent Service Management, March 23-25, 2009, Leipzig, Germany. GI, S. 19–31.

[DRG+10a] Delgado, A.; Ruiz, F.; García-Rodríguez De Guzmán, I.; Piattini, M. (2010): A Model-driven and Service-oriented framework for the business process improvement. In: Journal of Systems Integration (3), S. 45–55.

[DRG+10b] Delgado, A.; Ruiz, F.; García-Rodríguez De Guzmán, I.; Piattini, M. (2010): MINERVA: Model drIveN and sErvice oRiented Framework for the Continuous Business Process improVement and relAted Tools. In: Hutchison, D.; Kanade, T.;

Kittler, J.; Kleinberg, J. M.; Mattern, F.; Mitchell, J. C.; Naor, M.; Nierstrasz, O.; Pandu Rangan, C.; Steffen, B.; Sudan, M.; Terzopoulos, D.; Tygar, D.; Vardi, M. Y.; Weikum, G.; Dan, A.; Gittler, F.; Toumani, F. (Hrsg.): Service-Oriented Computing, ICSOC/ServiceWave 2009 Workshops. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, S. 456–466.

[DRG+11] Delgado, A.; Ruiz, F.; Guzmán, I. G.-R. de; Piattini, M. (2011): Business Process Service Oriented Methodology (BPSOM) with Service Generation in SoaML. In: Hutchison, D.; Kanade, T.; Kittler, J.; Kleinberg, J. M.; Mattern, F.; Mitchell, J. C.; Naor, M.; Nierstrasz, O.; Pandu Rangan, C.; Steffen, B.; Sudan, M.; Terzopoulos, D.; Tygar, D.; Vardi, M. Y.; Weikum, G.; Mouratidis, H.; Rolland, C. (Hrsg.): Advanced Information Systems Engineering. Springer, Berlin, Heidelberg, S. 672–680.

[DRR11] Dadam, P.; Reichert, M.; Rinderle-Ma, S. (2011): Prozessmanagementsysteme: Nur ein wenig Flexibilität wird nicht reichen. In: Informatik Spektrum 34 (4), S. 364–376.

[Dude07] Dudenredaktion (2007): Duden - Das große Fremdwörterbuch – Herkunft und Bedeutung der Fremdwörter. 4. Aufl., Dudenverlag, Mannheim.

[Dunc95] Duncan, N. B. (1995): Capturing Flexibility of Information Technology Infrastructure – A Study of Resource Characteristics and their Measure. In: Journal of Management Information Systems 12 (2), S. 37–57.

[DuSu05] Duerst, M.; Suignard, M. (2005): Internationalized Resource Identifiers (IRIs) – RFC 3987. URL: http://tools.ietf.org/html/rfc3987/ (Abruf: 25.04.2014).

[DVE10] Dominique Guinard; Vlad Trifa; Erik Wilde (2010): A resource oriented architecture for the Web of Things. In: Florian Michahelles; Jin Mitsugi (Hrsg.): Proceedings Internet of Things – IoT for a green Planet (IOT 2010), November, 29 –December, 1, 2010, Tokyo, Japan. IEEE, S. 1–8.

[DWR+13] Delgado, A.; Weber, B.; Ruiz, F.; Guzmán, I. G.-R. de; Piattini, M. (2013): Continuous Improvement of Business Processes Realized by Services Based on Execution Measurement. In: Maciaszek, L. A.; Zhang, K. (Hrsg.): Evaluation of Novel Approaches to Software Engineering. Springer, Berlin, Heidelberg, S. 64–81.

[EAA+04] Endrei, M.; Ang, J.; Arsanjani, A.; Chua, S.; Comte, P.; Krogdahl, P.; Luo, M.; Newling, T. (2004): Patterns: Service-Oriented Architecture and Web Services. IBM Redbooks.

[ECMA11] ECMA (2011): ECMAScript Language Specification – Standard ECMA-262. URL: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf (Abruf: 28.04.2014).

[ECMA13] ECMA (2013): The JSON Data Interchange Format – ECMA - 123:2009 - Standard ECMA-404. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (Abruf: 28.04.2014).

[ECP+13] Erl, T.; Carlyle, B.; Pautasso, C.; Balasubramanian, R. (2013): SOA with REST – Principles, Patterns & Constraints for Building Enterprise Solutions with REST. Prentice Hall, Upper Saddle River, NJ.

[Edli11] Edlich, S. (2011): NoSQL – Einstieg in die Welt nichtrelationaler Web-2.0-Datenbanken. Hanser, München.

[EMFC14] EMF Compare Community (2014): EMF Compare — Developer Guide. URL: http://www.eclipse.org/emf/compare/documentation/latest/developer/developer-guide.html (Abruf: 02.03.2015).

[Erl08a] Erl, T. (2008): Service-Oriented Architecture – Concepts, Technology, and Design. Prentice Hall, Upper Saddle River, NJ, Munich.

[Erl08b] Erl, T. (2008): SOA – Principles of Service Design. Prentice Hall, Upper Saddle River, NJ.

[Erl08c] Erl, T. (2008): Web Service Contract Design and Versioning for SOA. Prentice Hall, Upper Saddle River, NJ, Munich.

[ERM10] Eckert, J.; Repp, N.; Martin, W. (2010): SOA Check 2010 – Status Quo und Trends im Vergleich zum SOA Check 2007 bis 2009. S.A.R.L. Martin/TU Darmstadt/IT.

[ErSi08] Erickson, J.; Siau, K. (2008): Web Services, Service-Oriented Computing, and Service-Oriented Architecture. In: Journal of Database Management 19 (3), S. 42–54.

[Fers79] Ferstl, O. K. (1979): Konstruktion und Analyse von Simulationsmodellen. Hain, Königstein/Traunstein.

[FeSi06] Ferstl, O. K.; Sinz, E. J. (2006): Modeling of Business Systems Using SOM. In: Bernus, P.; Martins Kai; Schmidt, G. (Hrsg.): Handbook on Architectures of Information Systems. International Handbook of Information Systems. 2. Aufl., Springer, Berlin, S. 347–367.

[FeSi13] Ferstl, O. K.; Sinz, E. J. (2013): Grundlagen der Wirtschaftsinformatik. 7. Aufl., Oldenbourg, München.

[FeSi90] Ferstl, O. K.; Sinz, E. J. (1990): Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik 32 (6), S. 566–581.

[FeSi91] Ferstl, O. K.; Sinz, E. J. (1991): Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik 33 (6), S. 477–491.

[FeSi93] Ferstl, O. K.; Sinz, E. J. (1993): Geschäftsprozeßmodellierung. In: Wirtschaftsinformatik 35 (6), S. 589–592.

[FeSi94] Ferstl, O. K.; Sinz, E. J. (1994): Programmiermodell für objektorientierte, erweiterbare Anwendungssysteme – Technischer Report. Informationen zum Exponat (CeBIT'94).

[FeSi95] Ferstl, O. K.; Sinz, E. J. (1995): Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. In: Wirtschaftsinformatik 37 (3), S. 209–220.

[FeSi96] Ferstl, O. K.; Sinz, E. J. (1996): Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach. In: König, W. (Hrsg.): Distributed information systems in business. Springer, Berlin, S. 159–179.

[FeSi97] Ferstl, O. K.; Sinz, E. J. (1997): Flexible Organizations Through Object-oriented and Transaction-oriented Information Systems. In: Krallmann, H. (Hrsg.): Wirtschaftsinformatik ’97. Physica-Verlag HD, Heidelberg, S. 393–411.

[Fett09] Fettke, P. (2009): How Conceptual Modeling Is Used. In: Communications of the AIS (CAIS) (25), S. 571–592.

[FeWa03] Feldmann, M.; Wagner, R. (2003): Strukturieren mit Multitrees – Ein Fachkonzept zur verbesserten Navigation in Hypermedia. In: Wirtschaftsinformatik 45 (6), S. 589–598.

[FGM+99] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. (1999): Hypertext Transfer Protocol - HTTP/1.1 – RFC 2616. URL: http://www.ietf.org/rfc/rfc2616/ (Abruf: 22.04.2014).

[Fiel00] Fielding, R. T. (2000): Architectural styles and the design of network-based software architectures. PhD Thesis, Irvine.

[Fink13a] Fink, A. (2013): HTML. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[Fink13b] Fink, A. (2013): HTTP. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[Fisc14] Fischer, K. (2014): Computer-Reservierungs-System (CRS). In: Springer Gabler Verlag (Hrsg.): Gabler Wirtschaftslexikon.

[FlMa08] Flanagan, D.; Matsumoto, Y. (2008): Die Programmiersprache Ruby – Ruby 1.8 und 1.9. 1. Aufl. (deutsche Ausgabe), O'Reilly, Beijing.

[Fran02] Frank, U. (2002): Multi-Perspective Enterprise Modeling (MEMO): Conceptual Framework and Modeling Languages: Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS-35), Honolulu, 2002, S. 1258–1267.

[Fran03] Frankel, D. (2003): Model Driven Architecture – Applying MDA to enterprise computing. Wiley, Indianapolis, Ind.

[Fran98] Frank, U. (1998): Evaluating Modelling Languages: Relevant Issues, Epistemological Challenges and a Preliminary Research Framework. In: Arbeitsberichte des Instituts für Wirtschafts- und Verwaltungsinformatik, Universität Koblenz-Landau.

[Fris12b] Frisendal, T. (2012): Opportunity: NoSQL and Big Data. In: Frisendal, T. (Hrsg.): Design Thinking Business Analysis. Springer, Berlin, Heidelberg, S. 111–112.

[FrRu07] France, R.; Rumpe, B. (2007): Model-driven Development of Complex Software: A Research Roadmap. In: Briand, L. C.; Wolf, A. L. (Hrsg.): Future of Software Engineering. May 23–25, 2007, Minneapolis, Minnesota. IEEE Computer Society, Los Alamitos, CA, S. 37–54.

[FrRü12] Freund, J.; Rücker, B. (2012): Praxishandbuch BPMN 2.0. 3. Aufl., Hanser, München.

[FrLa03] Frank, U.; van Laak, B. (2003): Anforderungen an Sprachen zur Modellierung von Geschäftsprozessen. In: Arbeitsberichte des Instituts für Wirtschaftsinformatik, Universität Koblenz-Landau (34).

[FSF+14] Frank, U.; Strecker, S.; Fettke, P.; Brocke, J.; Becker, J.; Sinz, E. J. (2014): Das Forschungsfeld „Modellierung betrieblicher Informationssysteme“. In: Wirtschaftsinformatik 56 (1), S. 49–54.

[Gabr13] Gabriel, R. (2013): Datenbankmanagementsysteme. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[GHJ+94] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. (1994): Design Patterns – Elements of Reusable Object-Oriented Software. 31. Aufl., Addison-Wesley, Boston.

[GoAk03] Gordijn, J.; Akkermans, J. M. (2003): Value based requirements engineering: exploring innovative e-commerce idea. In: Requirements Engineering Journal 8 (2), S. 114–134.

[GoFi94] Gotel, O.; Finkelstein, C. W. (1994): An analysis of the requirements traceability problem: Proceedings of the First International Conference on Requirements Engineering, S. 94–101.

[Góme13] Gómez, Jorge Carlos Marx (2013): Serviceorientierte Architektur. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[GoPr08] Goodson, K.; Preotiuc-Pietro, R. (2008): Service Data Objects For Java Specification – Version 2.1.1 Draft, November 2008. Final Release Specification.
URL: https://www.jcp.org/en/jsr/detail?id=235 (Abruf: 28.09.2014).

[GoTo02] Gourley, D.; Totty, B. (2002): HTTP – The definitive guide. O'Reilly, Beijing.

[Grah08] Graham, I. (2008): Requirements modelling and specification for service oriented architecture. Wiley, Chichester.

[Grei03] Greiffenberg, S. (2003): Methoden als Theorien der Wirtschaftsinformatik. In: Uhr, W.; Esswein, W.; Schoop, E. (Hrsg.): Wirtschaftsinformatik 2003/Band II. Physica-Verlag HD, S. 947–967.

[Grei04] Greiffenberg, S. (2004): Methodenentwicklung in Wirtschaft und Verwaltung. Kovač, Hamburg.

[GrHo07] Gregorio, J.; Hora, B. de (2007): The Atom Publishing Protocol – RFC 5023. URL: http://www.ietf.org/rfc/rfc5023.txt (Abruf: 07.05.2014).

[Gron09] Gronback, R. C. (2009): Eclipse modeling project – A domain-specific language [(DSL)] toolkit. Addison-Wesley, Upper Saddle River, NJ.

[GrSh03] Greenfield, J.; Short, K. (2003): Software Factories – Assembling Applications with Patterns, Models, Frameworks and Tools. In: Crocker, R. (Hrsg.): Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. ACM, New York, NY, S. 16–27.

[GrSh04] Greenfield, J.; Short, K. (2004): Software factories – Assembling applications with patterns, models, frameworks, and tools. Wiley, Indianapolis, Ind.

[GTP07] Giner, P.; Torres, V.; Pelechano, V. (2007): Bridging the Gap between BPMN and WS-BPEL – M2M Transformations in Practice: Proceedings of the 3rd International Workshop on Model-Driven Web Engineering (MDWE 2007), Como, Italy.

[GuLa10] Gu, Q.; Lago, P. (2010): Service Identification Methods: A Systematic Literature Review. In: Di Nitto, E.; Yahyapour, R. (Hrsg.): Towards a service-based internet. Proceedings of the Third European Conference, ServiceWave 2010, December 13-15, Ghent, Belgium. Springer, Berlin, S. 37-50.

[Gula13] Gulabani, S. (2013): Developing RESTful Web Services with Jersey 2.0 – Create RESTful web services smoothly using the robust Jersey 2.0 and JAX-RS APIs. Packt Pub, Birmingham, UK.

[Guld09] Gulden, J. (2009): Minimal invasive generative Entwicklung von Modellierungswerkzeugen. In: Fischer, S.; Maehle, E.; Reischuk, R. (Hrsg.): Tagungsband der Konferenz INFORMATIK 2009. LNI Proceedings, Beitrag 270.

[Hadl09] Hadley, M. (2009): Web Application Description Language – W3C Member Submission 31. August 2009. URL: http://www.w3.org/Submission/wadl/ (Abruf: 28.09.2014).

[Häre14] Härer, F. (2014): Vom SOM-Geschäftsprozessmodell zum Softwareartefakt – modellgetriebene Systementwicklung mit dem Eclipse Modeling Framework. Masterarbeit, Bamberg.

[Hart11] Hartmann, B. (2011): Enterprise Architecture as an Instrument of Strategic Control. In: Nüttgens, M.; Thomas, O.; Weber, B. (Hrsg.): Enterprise Modelling and Information Systems Architectures (EMISA 2011). GI, Bonn, S. 9–22.

[HaTa06] Hailpern, B.; Tarr, P. (2006): Model-driven development: The good, the bad, and the ugly. In: IBM Systems Journal 45 (3), S. 451–461.

[HaWo12] Hartmann, B.; Wolf, M. (2012): Erweiterung einer Geschäftsprozessmodellierungssprache zur Stärkung der strategischen Ausrichtung von Geschäftsprozessen. In: Sinz, E. J.; Schürr, A. (Hrsg.): Modellierung 2012 (MOD 2012). 14.-16. März, 2012, Bamberg, Deutschland. GI, Bonn, S. 235–250.

[HdC09] Higashino, W. A.; de Toledo, M. Beatriz Felgar; Capretz, Miriam A. M. (2009): REST and Resource-Oriented Architecture. In: Alt, R.; Fähnrich, K.-P., Franczyk, B. (Hrsg.): Proceedings of the First International Symposium on Services Science (ISSS'09). Logos, Berlin, S. 161–171.

[HGN10] Hogrebe, F.; Gehrke, N.; Nüttgens, M. (2010): Gebrauchstauglichkeit semiformaler Modellierungssprachen für das Anforderungsmanagement – Untersuchungsrahmen, Anwendungsfall und experimentelle Evaluation mittels Blickbewegungsregistrierung. In: Engels, G.; Karagiannis, D.; Mayr, H. C. (Hrsg.): Modellierung 2010 (MOD 2010). 24. - 26. März 2010, Klagenfurt, Österreich. GI, Bonn, S. 31–48.

[HKR+08] Hitzler, P.; Krötzsch, M.; Rudolph, S.; Sure, Y. (2008): Semantic Web – Grundlagen. Springer, Berlin.

[Hohp07] Hohpe G.: Konversationen zwischen lose gekoppelten Services. In: Starke, G.; Tilkov, S. (Hrsg.): SOA-Expertenwissen – Methoden, Konzepte und Praxis serviceorientierter Architekturen. dpunkt, Heidelberg, S. 439-446.

[Hopf89] Hopfmann, L. (1989): Flexibilität im Produktionsbereich – Ein dynamisches Modell zur Analyse und Bewertung von Flexibilitätspotentialen. P. Lang, Frankfurt am Main, New York.

[HPF10] Hahn, C.; Panfilenko, D.; Fischer, K. (2010): A model-driven approach to close the gap between business requirements and agent-based execution. In: Proceedings of the 4th Workshop on Agent-based Technologies and applications for enterprise interoperability, Toronto, Canada, S. 13–24.

[HRO+10] Holschke, O.; Rake, J.; Offermann, P.; Bub, U. (2010): Steigerung der Softwareflexibilität bei Geschäftsprozessänderungen. In: Wirtschaftsinformatik 52 (1), S. 3–15.

[HSW98] Hammel, C.; Schlitt, M.; Wolf, S. (1998): Pattern-basierte Konstruktion von Unternehmensmodellen. In: Informationssystem-Architekturen - Rundbrief der GI-Fachgruppe 5.2 der Gesellschaft für Informatik (1), S. 22–37.

[HTW13] Hartmann, B.; Teusch, A.; Wolf, M. (2013): Spezifikation von funktionalen und nichtfunktionalen Systemanforderungen auf Basis von Geschäftsprozessmodellen. In: Wirtschaftsinformatik Proceedings 2013, S. 1293–1307.
URL: http://aisel.aisnet.org/wi2013/81 (Abruf: 2.10.2014)

[IANA14] IANA (2014): Media Types. URL: http://www.iana.org/assignments/media-types/ (Abruf: 22.04.2014).

[III11] ISO; IEC; IEEE (2011): ISO/IEC/IEEE 42010:2011 International Standard – Systems and software engineering — Architecture description.
URL: http://www.iso-architecture.org/ieee-1471/ (Abruf: 11.04.2014).

[JAB+06] Jouault, F.; Allilaire, F.; Bézivin, J.; Kurtev, I.; Valduriez, P. (2006): ATL: a QVT-like transformation language. In: Tarr, P.; Cook, W. R. (Hrsg.): Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (OOPSLA'06), S. 719–720.

[Jack91] Jackson, J. (1991): A Keyphrase Based Traceability Scheme. In: IEEE Colloquium on Tools and Techniques for Maintaining Traceability During Design (180), S. 2/1–2/4.

[Jaco67] Jacob, H. (1967): Flexibilitätsüberlegungen in der Investitionsrechnung. In: Zeitschrift für Betriebswirtschaft 37, S. 1–34.

[Jaco74] Jacob, H. (1974): Unsicherheit und Flexibilität – Zur Theorie der Planung bei Unsicherheit. In: Zeitschrift für Betriebswirtschaft 44, S. (1/3) 229-326, (2/3) 403-448, (3/3) 505-526.

[JFJ+09] Jürck, C.; Förtsch, T.; Jahn, B.; Ulbrich-vom Ende, A. (2009): Einsatz von Recommender-Systemen zur personalisierten Informationsversorgung im Standardberichtswesen von Data-Warehouse-Systemen. In: Wirtschaftsinformatik Proceedings 2009. Paper 114.

[Josu08] Josuttis, N. M. (2008): SOA in der Praxis – System-Design für verteilte Geschäftsprozesse. dpunkt, Heidelberg.

[Josu09] Josuttis, N. (2009): Erläuterungen zum SOA-Manifest – Präambel, Wertesystem und Prinzipien. URL: http://www.soa-manifest.de/kommentar.html
(Abruf: 17.04.2014).

[KaBl00] Kaluza, B.; Blecker, T. (2000): Wettbewerbsstrategien - Markt- und ressourcenorientierte Sicht der strategischen Führung - Konzepte - Gestaltung – Umsetzungen. TCW, München.

[KaBl05a] Kaluza, B.; Blecker, T. (2005): Erfolgsfaktor Flexibilität – Strategien und Konzepte für wandlungsfähige Unternehmen. Schmidt, Berlin.

[KaBl05b] Kaluza, B.; Blecker, T. (2005): Flexibilität – State of the Art und Entwicklungstrends. In: Kaluza, B.; Blecker, T. (Hrsg.): Erfolgsfaktor Flexibilität. Strategien und Konzepte für wandlungsfähige Unternehmen. Schmidt, Berlin, S. 1–28.

[KaKr08] Kaczmarek, T.; Krzysztof, W. (2008): Hype over Service Oriented Architecture Continues. In: Wirtschaftsinformatik 50 (1), S. 52–59.

[Kalv31] Kalveram, W. (1931): Elastizität und Betriebsführung. In: Zeitschrift für Betriebswirtschaft 1 (8), S. 705–711.

[KaMe13] Kalali, M.; Mehta, B. (2013): Developing RESTful services with JAX-RS 2.0, WebSockets, and JSON. Packt Pub, Birmingham, U.K.

[Kara13] Karagiannis, D. (2013): CASE-Tools. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[KBS05] Krafzig, D.; Banke, K.; Slama, D. (2005): Enterprise SOA – Service-oriented architecture best practices. Prentice Hall Professional Technical Reference, Indianapolis, IN.

[KDP+09] Kolovos, D. S.; Di Ruscio, D.; Pierantonio, A.; Paige, R. F. (2009): Different models for model matching – An analysis of approaches to support model differen-

cing: ICSE Workshop on Comparison and Versioning of Software Models (CVSM'09), Vancouver, BC, Canada, S. 1–6.

[KeEi13] Kemper, A.; Eickler, A. (2013): Datenbanksysteme – Eine Einführung. 9. Aufl., Oldenbourg, München.

[Kell07] Keller, W. (2007): SOA-Governance – SOA langfristig durchsetzen und managen. In: Starke, G.; Tilkov, S. (Hrsg.): SOA-Expertenwissen – Methoden, Konzepte und Praxis serviceorientierter Architekturen. dpunkt, Heidelberg, S. 289–307.

[Kelt07] Kelter, U. (2007): Begriffliche Grundlagen von Modelldifferenzen – Positionspapier. In: Workshop „Vergleich und Versionierung von UML-Modellen", SE2007, Hamburg, Deutschland.

[Kent02] Kent, S. (2002): Model Driven Engineering. In: Butler, M.; Petre, L.; Sere, K. (Hrsg.): Integrated Formal Methods. Springer, Berlin, Heidelberg, S. 286–298.

[KeTo08] Kelly, S.; Tolvanen, J.-P. (2008): Domain-specific modeling – Enabling full code generation. Wiley, Hoboken, NJ.

[KGV08] Kopecký, J.; Gomadam, K.; Vitvar, T. (2008): hRESTS: An HTML Microformat for Describing RESTful Web Services. In: Jain, L. C.; Li, Y. (Hrsg.): Proceeding of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Sydney, Australia. IEEE Computer Society, Los Alamitos, Calif, S. 619–625.

[KJL09] Kumar, S.; Johnson, K. L.; Lai, S. T. (2009): Performance improvement possibilities within the US airline industry. In: International Journal of Productivity and Performance Management 58 (7), S. 694–717.

[KRR+10] Kantor, P.; Ricci, F.; Rokach, L.; Shapira, B. (Hrsg.): Recommender Systems Handbook. Springer, Berlin (2010).

[Klum02] Klumb, M. J. (2002): Organisationale Flexibilität und Marktstrukturen – Moderne Organisationsformen und ihre Rolle im globalen Wettbewerb. Deutscher Universitätsverlag, Wiesbaden.

[KNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W. (1992): Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK). In: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Saarbrücken.

[Kosi76] Kosiol, E. (1976): Organisation der Unternehmung. 2. Aufl., Gabler, Wiesbaden.

[KrSi11] Krücke, A.; Sinz, E. J. (2011): Entwurf partieller SOA auf der Grundlage von Geschäftsprozessmodellen. In: Sinz, E. J.; Bartmann, D.; Bodendorf, F.; Ferstl, O. K. (Hrsg.): Dienstorientierte IT-Systeme für hochflexible Geschäftsprozesse. University of Bamberg Press, Bamberg, S. 287–312.

[KSH08] Kohnke, O.; Scheffler, T.; Hock, C. (2008): SOA-Governance – Ein Ansatz zum Management serviceorientierter Architekturen. In: Wirtschaftsinformatik 50 (5), S. 408–412.

[Kuhr13] Kuhrmann, M. (2013): Wasserfallmodell. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[Kurb90] Kurbel, K. (Hrsg.): Handbuch Wirtschaftsinformatik. Poeschel, Stuttgart (1990).

[KWB04] Kleppe, A.; Warmer, J. B.; Bast, W. (2004): MDA explained – The model driven architecture: practice and promise. Pearson Education, Delhi.

[LaCr13] Lake, P.; Crowther, P. (2013): NoSQL Databases. In: Lake, P.; Crowther, P. (Hrsg.): Concise Guide to Databases. Springer, London, S. 97–134.

[LaGü12] Lanthaler, M.; Gütl, C. (2012): On Using JSON-LD to Create Evolvable RESTful Services. In: Alarcon, R.; Pautasso, C.; Wilde, E. (Hrsg.): Proceedings of the Third International Workshop on RESTful Design (WS-REST 2012), April 17, 2012, Lyon, France, S. 25–32.

[LaGü13] Lanthaler, M.; Gütl, C. (2013): Creating 3rd Generation Web APIs with JSON-LD and Hydra. In: Pautasso, C.; Wilde, E.; Alarcón, R. (Hrsg.): Proceedings of the Forth International Workshop in RESTful Design (WS-REST 2013), May 13–17, 2013, Rio de Janeiro, Brazil, S. 35–37.

[Lete02] Letelier, P. (2002): A Framework for Requirements Traceability in UML-based Projects. In: Spanoudakis, G.; Zisman, A.; Perez-Minana, E. (Hrsg.): Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'02), Edinburgh, Scotland, S. 30–41.

[LGS07] Lathem, J.; Gomadam, K.; Sheth, A. (2007): SA-REST and (S)mashups: Adding Semantics to RESTful Services: Proceedings of the International Conference on Semantic Computing (ICSC 2007), S. 469–476.

[LKS06] Laitkorpi, M.; Koskinen, J.; Systä, T. (2006): A UML-based Approach for Abstracting Application Interfaces to REST-like Services: Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06), October 23-27, 2006, Benevento, Italy. IEEE, Los Alamitos, Calif, S. 134–146.

[LME08] Lucchi, R.; Millot, M.; Elfers, C. (2008): Resource Oriented Architecture and REST. URL: http://inspire.ec.europa.eu/reports/ImplementingRules/network/Resource_orientated_architecture_and_REST.pdf (Abruf: 28.04.2014).

[LSS09] Laitkorpi, M.; Selonen, P.; Systa, T. (2009): Towards a Model-Driven Process for Designing ReSTful Web Services. In: Proceedings of the 2009 IEEE International Conference on Web Services (ICWS 2009), July 6–10, 2009, Los Angeles, CA. IEEE Computer Society, Washington, DC,S. 173–180.

[MaBu90] Mandelbaum, M.; Buzacott, J. A. (1990): Flexibility and decision making. In: European Journal of Operational Research 44, S. 17–27.

[MAC06] Marcos, E.; Acuña, C. J.; Cuesta, C. E. (2006): Integrating Software Architecture into a MDA Framework. In: Hutchison, D.; Kanade, T.; Kittler, J.; Kleinberg, J. M.; Mattern, F.; Mitchell, J. C.; Naor, M.; Nierstrasz, O.; Pandu Rangan, C.; Steffen, B.; Sudan, M.; Terzopoulos, D.; Tygar, D.; Vardi, M. Y.; Weikum, G.; Gruhn, V.; Oquendo, F. (Hrsg.): Software Architecture. Springer, Berlin, Heidelberg, S. 127–143.

[Mahl07] Mahlberg, M. (2007): SOA »von unten«. In: Starke, G.; Tilkov, S. (Hrsg.): SOA-Expertenwissen – Methoden, Konzepte und Praxis serviceorientierter Architekturen. dpunkt, Heidelberg, S. 169–186.

[Mali97] Malischewski, C. (1997): Generierung von Spezifikationen betrieblicher Anwendungssysteme auf der Basis von Geschäftsprozeßmodellen. Shaker, Aachen.

[MaSm95] March, S. T.; Smith, G. F. (1995): Design and natural science research on information technology. In: Decision Support Systems 15 (4), S. 251–266.

[MCB+11] Manyika, J.; Chui, M.; Brown, B.; Bughin, J.; Dobbs, R.; Roxburgh, C.; Hung Byers; Angela (2011): Big data: The next frontier for innovation, competition, and productivity. URL: http://www.mckinsey.com/~/media/McKinsey/dotcom/Insig-

hts%20and%20pubs/MGI/Research/Technology%20and%20Innovation/Big%20Data/MGI_big_data_full_report.ashx (Abruf: 28.04.2014).

[MCV03] Marcos, E.; Castro, V. de; Vela, B. (2003): Representing Web Services with UML: A Case Study. In: Goos, G.; Hartmanis, J.; van Leeuwen, J.; Orlowska, M. E.; Weerawarana, S.; Papazoglou, M. P.; Yang, J. (Hrsg.): Service-Oriented Computing - ICSOC 2003. Springer, Berlin, Heidelberg, S. 17–27.

[MeDe08] Mens, T.; Demeyer, S. (2008): Software evolution. Springer, New York, London.

[Meff68] Meffert, H. (1968): Die Flexibilität in betriebswirtschaftlichen Entscheidungen. Habilitationsschrift, München.

[Meff85] Meffert, H. (1985): Größere Flexibilität als Unternehmenskonzept. In: Zeitschrift für betriebswirtschaftliche Forschung 38 (2), S. 121–137.

[Melz10] Melzer, I. (2010): Service-orientierte Architekturen mit Web Services – Konzepte - Standards - Praxis. 4. Aufl., Spektrum Akademischer Verlag, Heidelberg, Neckar.

[MGP+08] Meliá, S.; Gómez, J.; Pérez, S.; Díaz, O. (2008): A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA: Proceedings of the 8th International Conference on Web Engineering (ICWE'08), Yorktown Heights, NJ, S. 13–23.

[MiLa07] Mitra, N.; Lafon, Y. (2007): SOAP Version 1.2 Part 0 – Primer Specification - W3C Recommendation. URL: http://www.w3.org/TR/2007/REC-soap12-part0-20070_427/ (Abruf: 14.04.2014).

[Mint05] Mintert, S. (2005): Implementierung von Webservices – REST vs. SOAP. In: Wirtschaftsinformatik 47 (1), S. 63–65.

[Mock87a] Mockapetris, P. (1987a): Domain Names - Implementation and Specification – RFC 1035. URL: http://tools.ietf.org/html/rfc1035/ (Abruf: 25.04.2014).

[Mock87b] Mockapetris, P. (1987b): Domain Names - Concepts and Facilities – RFC 1034. URL: http://tools.ietf.org/html/rfc1034/ (Abruf: 25.04.2014).

[Moos09] Moos, C. (2009): Komplexität, Flexibilität und Erfolg als Herausforderungen marktorientierter Fertigungsstrategien. In: Milling, P.; Strohhecker, J.; Grössler, A. (Hrsg.): Strategisches und operatives Produktionsmanagement. Empirie und Simulation. Gabler-Springer, Wiesbaden, S. 47–70.

[MPD10] Maleshkova, M.; Pedrinaci, C.; Domingue, J. (2010): Investigating Web APIs on the World Wide Web. In: Proceeding of the 2010 8th IEEE European Conference on Web Services (ECOWS'10), S. 107–114.

[MRP06] Maeder, P.; Riebisch, M.; Philippow, I. (2006): Traceability for Managing Evolutionary Change – A Roadmap. In: Dosch, W.; Perrizo, W. (Hrsg.): Proceedings of the 15th International Conference on Software Engineering and Data Engineering (SEDE-2006), July 6–8, 2006, Los Angeles, Calif., S. 1–8.

[MRS09] Manning, C. D.; Raghavan, P.; Schütze, H. (2009): An Introduction to Information Retrieval (Online-Version (Draft), 1 April 2009). Cambridge University Press, Cambridge, England.

[MRS11] Montangero, C.; Reiff-Marganiec, S.; Semini, L. (2011): Model-Driven Development of Adaptable Service-Oriented Business Processes. In: Hutchison, D.; Kanade, T.; Kittler, J.; Kleinberg, J. M.; Mattern, F.; Mitchell, J. C.; Naor, M.; Nierstrasz, O.; Pandu Rangan, C.; Steffen, B.; Sudan, M.; Terzopoulos, D.; Tygar, D.; Vardi, M. Y.; Weikum, G.; Wirsing, M.; Hölzl, M. (Hrsg.): Rigorous Software Engineering for Service-Oriented Systems. Springer, Berlin, Heidelberg, S. 115–132.

[Müll07] Müller, T. (2007): Der Weg zum guten Service. In: Starke, G.; Tilkov, S. (Hrsg.): SOA-Expertenwissen – Methoden, Konzepte und Praxis serviceorientierter Architekturen. dpunkt, Heidelberg, S. 141–159.

[MüWe12] Müller, B.; Wehr, H. (2012): Java Persistence API 2 – Hibernate, EclipseLink, OpenJPA und Erweiterungen. Hanser, München.

[NAK14] Nimis, J.; Armbruster, M.; Kammerer, M. (2014): Zukunftsfähiges Datenmanagement durch hybride Lösungen – Ein Entwurfsmusterkatalog zur Integration von SQL- und NoSQL-Datenbanken. In: Jähnert, J.; Förster, C. (Hrsg.): Technologien für digitale Innovationen. Springer, Wiesbaden, S. 19–42.

[Neum13] Neumann, G. (2013): XML. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[Niel99] Nielsen, J. (1999): User interface directions for the Web. In: Communications of the ACM 42 (1), S. 65–72.

[NoSa05] Nottingham, M.; Sayra, R. (2005): The Atom Syndication Format – RFC 4287. URL: http://www.ietf.org/rfc/rfc4287.txt (Abruf: 07.05.2014).

[NSA14] Neumann, G.; Sobernig, S.; Aram, M. (2014): Evolutionäre betriebliche Informationssysteme – Perspektiven und Herausforderungen einer neuen Generation von Informationssystemen. In: Wirtschaftsinformatik 56 (1), S. 41–47.

[OASI07] OASIS (2007): Web Services Business Process Execution Language Version 2.0 – OASIS Standard. URL: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html (Abruf: 01.06.2014).

[OASI12] OASIS (2012): Reference Architecture Foundation for Service Oriented Architecture Version 1.0. Committee Specification 01. URL: http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/cs01/soa-ra-v1.0-cs01.pdf (Abruf: 28.06.2013).

[OLA+12] Ormeno, E.; Lund, M.; Aballay, L.; Aciar, S. (2012): An UML profile for modeling RESTful services. In: Proceedings of the 13th Argentine Symposium on Software Engineering (ASSE 2012), S. 119–133.

[OlNe06] Oldevik, J.; Neple, T. (2006): Traceability in Model to Text Transformations. In: Proceedings of the 2nd European conference on Model driven architecture (ECMDA 2006) - Traceability Workshop, Bilbao, S. 144–156.

[OlOl07] Olsen, G. K.; Oldevik, J. (2007): Scenarios of traceability in model to text transformations. In: Akehurst, D. H.; Vogel, R.; Paige, R. F. (Hrsg.): Model driven architecture. Foundations and applications 3rd European conference (ECMDA-FA 2007), June 11–15, 2007, Haifa, Israel, Proceedings. Springer, Berlin, S. 144–156.

[OMG01] OMG (2001): Model Driven Architecture (MDA) – Document number ormsc/2001-07-01. URL: http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01/
(Abruf: 15.04.2014).

[OMG03] OMG (2003): MDA Guide Version 1.0.1 – Document Number omg/2003-06-01. URL: http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf (Abruf: 02.10.2014).

[OMG11a] OMG (2011): Business Process Model and Notation (BPMN) 2.0. URL: http://www.omg.org/spec/BPMN/2.0/ (Abruf: 11.11.2014).

[OMG11b] OMG (2011): Meta Object Facility (MOF) 2.0 Query/View/ Transformation Specification (QVT) – Version 1.1 - January 2011.
URL: http://www.omg.org/spec/QVT/1.1/ (Abruf: 20.08.2014).

[OMG11c] OMG (2011): Unified Modeling Language (UML) Infrastructure – Version 2.4.1. URL: http://www.omg.org/spec/UML/2.4.1/ (Abruf: 01.04.2014).

[OMG12] OMG (2012): Service oriented architecture Modeling Language (SoaML) Specification – Version 1.0.1. URL: http://www.omg.org/spec/SoaML/1.0.1/ (Abruf: 01.12.2014).

[OMG13a] OMG (2013): Meta Object Facility (MOF) Core Specification – Version 2.4.1. URL: http://www.omg.org/spec/MOF/2.4.1/ (Abruf: 15.04.2014).

[OMG13b] OMG (2013): XMI – Version 2.4.1. URL: http://www.omg.org/spec/XMI/2.4.1/ (Abruf: 15.04.2014).

[OMG14] OMG (2014): Object Constraint Language – Version 2.4.
URL: http://www.omg.org/spec/OCL/2.4/ (Abruf: 15.04.2014).

[Over07] Overdick, H. (2007): The Resource-Oriented Architecture: Proceedings of the IEEE Congress on Services (Services'07), S. 340–347.

[Over08] Overdick, H. (2008): Towards Resource-Oriented BPEL. In: Cesare Pautasso; Thomas Gschwind (Hrsg.): Proceedings of the 2nd ECOWS07 Workshop on Emerging Web Services Technology (WEWST 2007), November 26, 2007, Halle (Saale), Germany. CEUR-WS.org Verlag.

[ÖWB10] Österle, H.; Winter, R.; Brenner, W. (2010): Gestaltungsorientierte Wirtschaftsinformatik – Ein Plädoyer für Rigor und Relevanz. Infowerk, Nürnberg.

[PaHe07] Papazoglou, M. P.; Heuvel, W.-J. (2007): Service oriented architectures: approaches, technologies and research issues. In: The VLDB Journal 16 (3), S. 389–415.

[Papa08] Papazoglou, M. P. (2008): Web services principles and technology. Pearson, Harlow, München.

[PaPa13] Panziera, L.; Paoli de, F. (2013): A Framework for Self-descriptive RESTful Services. In: Pautasso, C.; Wilde, E.; Alarcón, R. (Hrsg.): Proceedings of the Fourth International Workshop in RESTful Design (WS-REST 2013), May 13–17, 2013, Rio de Janeiro, Brazil, S. 1407–1414.

[Paut08] Pautasso, C. (2008): BPEL for REST. In: Dumas, M.; Reichert, M.; Shan, M.-C. (Hrsg.): Business Process Management. Springer, Berlin, Heidelberg, S. 278–293.

[Paut09a] Pautasso, C. (2009): Composing RESTful Services with JOpera. In: Bergel, A.; Fabry, J. (Hrsg.): Proceedings of the 8th International Conference on Software Composition (SC'09). Springer, Berlin, Heidelberg, S. 142–159.

[Paut09b] Pautasso, C. (2009): RESTful Web service composition with BPEL for REST. In: Data & Knowledge Engineering 68 (9), S. 851–866.

[Paut11] Pautasso, C. (2011): BPMN for REST. In: Dijkman, R.; Hofstetter, J.; Koehler, J. (Hrsg.): Business Process Model and Notation. Springer, Berlin, S. 74–87.

[Paut14] Pautasso, C. (2014): RESTful Web Services: Principles, Patterns, Emerging Technologies. In: Bouguettaya, A.; Sheng, Q. Z.; Daniel, F. (Hrsg.): Web Services Foundations. Springer, New York, NY, S. 31–51.

[PaWi11] Pautasso, C.; Wilde, E. (2011): Push-Enabling RESTful Business Processes. In: Hutchison, D.; Kanade, T.; Kittler, J.; Kleinberg, J. M.; Mattern, F.; Mitchell, J. C.; Naor, M.; Nierstrasz, O.; Pandu Rangan, C.; Steffen, B.; Sudan, M.; Terzopoulos, D.; Tygar, D.; Vardi, M. Y.; Weikum, G.; Pallis, G.; Jmaiel, M.; Charfi, A.; Graupner, S.; Karabulut, Y.; Guinea, S.; Rosenberg, F.; Sheng, Q. Z.; Pautasso, C.; Mokhtar, S. (Hrsg.): Service-Oriented Computing - ICSOC 2011 Workshops. Springer, Berlin, Heidelberg, S. 32–46.

[PDM+11] Perez, S.; Durao, F.; Melia, S.; Dolog, P.; Diaz, O. (2011): RESTful, Resource-Oriented Architectures: A Model-Driven Approach. In: Proceedings of Web information systems engineering (WISE 10), December 12–14, 2010, Hong Kong, China. Springer, Berlin, S. 282–294.

[PeMe06] Petrasch, R.; Meimberg, O. (2006): Model Driven Architecture – Eine praxisorientierte Einführung in die MDA. dpunkt, Heidelberg.

[PfAt10] Pfleeger, S. L.; Atlee, J. M. (2010): Software engineering – Theory and practice. 4. Aufl., Pearson, Boston.

[Pohl08] Pohl, K. (2008): Requirements engineering – Grundlagen, Prinzipien, Techniken. 2. Aufl., dpunkt, Heidelberg.

[Poma12] Pomaska, G. (2012): Webseiten-Programmierung – Sprachen, Werkzeuge, Entwicklung. Springer Vieweg, Wiesbaden.

[PoRa11] Porres, I.; Rauf, I. (2011): Modeling behavioral RESTful web service interfaces in UML. In: Chu, W.; Wong, W. E.; Palakal, M. J.; Hung, C.-C. (Hrsg.): Proceedings of the 2011 ACM Symposium on Applied Computing (SAC 11), S. 1598–1605.

[Pres10] Pressman, R. S. (2010): Software engineering – A practitioner's approach. 7. Aufl., McGraw-Hill, Boston.

[PüSi10] Pütz, C.; Sinz, E. J. (2010): Modellgetriebene Ableitung von BPMN-Workflowschemata aus SOM-Geschäftsprozessen. In: Engels, G.; Karagiannis, D.; Mayr, H. C. (Hrsg.): Modellierung 2010 (MOD 2010). 24.–26. März, 2010, Klagenfurt, Österreich. GI, Bonn, S. 253–268.

[PWS+10] Parastatidis, S.; Webber, J.; Silveira, G.; Robinson, I. S. (2010): The role of hypermedia in distributed system development. In: Pautasso, C.; Wilde, E.; Marinos, A. (Hrsg.): Proceedings of the First International Workshop on RESTful Design (WS-REST 2010), April 26, 2010, Raleigh, NC, S. 16–22.

[PZL08] Pautasso, C.; Zimmermann, O.; Leymann, F. (2008): RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. In: Huai, J. (Hrsg.): Proceedings of the 17th International Conference on World Wide Web (WWW'08). ACM Press, New York, N.Y, S. 805–814.

[RaBe06] Rahm, E.; Bernstein, P. A. (2006): An online bibliography on schema evolution. In: ACM SIGMOD Record 35 (4), S. 30–31.

[RAL+06] Rizzi, S.; Abelló, A.; Lechtenbörger, J.; Trujillo, J. (2006): Research in data warehouse modeling and design: dead or alive? In: Song, I.-Y.; Vassiliadis, P. (Hrsg.): Proceedings of the 9th ACM international workshop on Data warehousing and OLAP. ACM, New York, NY, S. 3–10.

[RaSc02] Rautenstrauch, C.; Schulze, T. (2002): Informatik für Wirtschaftswissenschaftler und Wirtschaftsinformatiker. Springer, Berlin.

[RCD+08] Rosenberg, F.; Curbera, F.; Duftler, M. J.; Khalaf, R. (2008): Composing RESTful Services and Collaborative Workflows: A Lightweight Approach. In: IEEE Internet Computing 12 (5), S. 24–31.

[RDS07] Ramollari, E.; Dranidis, D.; Simons, A.J.H. (2007): A Survey of Service Oriented Development Methodologies. In: Gorton, S; Solanki, M; Reiff-Marganiec, S (Hrsg.): Proceedings of 2nd Young Researchers' Workshop on Service Oriented Computing. Leicester, U.K. S. 75-80.

[Reen79] Reenskaug, T. (1979): MVC: XEROX PARC 1978-79. URL: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html (Abruf: 05.06.2014).

[ReFr05] Reddy, R.; France, R. (2005): Model Composition - A Signature-Based Approach. In: Proceedings of Aspect Oriented Modeling (AOM) Workshop, October, 2005, Montego, Jamaica.

[Reis10] Reisig, W. (2010): Petrinetze – Modellierungstechnik, Analysemethoden, Fallstudien. Vieweg + Teubner, Wiesbaden.

[Remm97] Remme, M. (1997): Konstruktion von Geschäftsprozessen – Ein modellgestützter Ansatz durch Montage generischer Prozeßpartikel. Gabler, Wiesbaden.

[Rest12] RestDoc.org (2012): RestDoc - Documenting REST APIs. Version 1. URL: http://www.restdoc.org/spec.html (Abruf: 15.07.2014).

[Rick07] Ricken, J. (2007): Top-Down Modeling Methodology for Model-Driven SOA Construction. In: Meersman, R.; Tari, Z.; Herrero, P. (Hrsg.): On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops. Springer, Berlin, Heidelberg, S. 323–332.

[RiRu07] Richardson, L.; Ruby, S. (2007): RESTful web services. O'Reilly, Farnham.

[RLJ99] Raggett, D.; Le Hors, A.; Jacobs, I. (1999): HTML 4.01 Specification – W3C Recommendation 24 December 1999. URL: http://www.w3.org/TR/1999/REC-html401-19991224 (Abruf: 24.04.2014).

[Romm08] Rommelspacher, J. (2008): Ereignisgetriebene Architekturen. In: Wirtschaftsinformatik 50 (4), S. 314–317.

[Roth12] Roth, M. (2012): Modellgetriebene Entwicklung Rails-basierter Web Anwendungen auf Basis von SOM-Anwendungssystemspezifikationen. Bachelorarbeit, Bamberg.

[RRS+06] Rahmani, A. T.; Rafe, V.; Sedighian, S.; Abbaspour, A. (2006): An MDA-Based Modeling and Design of Service Oriented Architecture. In: Hutchison, D.; Kanade, T.; Kittler, J.; Kleinberg, J. M.; Mattern, F.; Mitchell, J. C.; Naor, M.; Nierstrasz, O.; Pandu Rangan, C.; Steffen, B.; Sudan, M.; Terzopoulos, D.; Tygar, D.; Vardi, M. Y.; Weikum, G.; Alexandrov, V. N.; Albada, G. D.; Sloot, P. M. A.; Dongarra, J. (Hrsg.): Computational Science – ICCS 2006. Springer, Berlin, Heidelberg, S. 578–585.

[RRS+10] Rauf, I.; Ruokonen, A.; Systa, T.; Porres, I. (2010): Modeling a Composite RESTful Web Service with UML. In: Gorton, I.; Babar, M. A.; Cuesta, C. E. (Hrsg.): Proceedings of the Fourth European Conference on Software Architecture (ECSA'10). ACM, New York, S. 253–260.

[RTH11] Ruby, S.; Thomas, D.; Hansson, D. H. (2011): Agile Web Development with Rails. 4. Aufl., The Pragmatic Bookshelf, Raleigh, NC.

[Rupp09] Rupp, C. (2009): Requirements-Engineering und -Management – Professionelle, iterative Anforderungsanalyse für die Praxis. 5. Aufl., Hanser, München, Wien.

[Russ08] Russell, C. (2008): Bridging the Object-Relational Divide. In: Queue 6 (3), S. 18-28.

[SBB+11] Sinz, E. J.; Bartmann, D.; Bodendorf, F.; Ferstl, O. K. (2011): Dienstorientierte IT-Systeme für hochflexible Geschäftsprozesse. University of Bamberg Press, Bamberg.

[SBP+08] Steinberg, D.; Budinsky, F.; Paternostro, M.; Merks, E. (2008): EMF - Eclipse Modeling Framework. 2. Aufl., Addison-Wesley, Upper Saddle River, NJ, München.

[Sche01] Scheer, A.-W. (2001): ARIS - Modellierungsmethoden, Metamodelle, Anwendungen. 4. Aufl., Springer, Berlin.

[Sche02] Scheer, A.-W. (2002): ARIS - vom Geschäftsprozess zum Anwendungssystem. 4. Aufl., Springer, Berlin.

[Schl04] Schlitt, M. (2004): Grundlagen und Methoden für Interpretation und Konstruktion von Informationssystemmodellen – Fundamentals and methods of interpreting and constructing information system models. Dissertation, Bamberg.

[Schm06] Schmidt, D. C. (2006): Guest Editor's Introduction: Model-Driven Engineering. In: IEEE Computer 39 (2), S. 25–31.

[Schm07] Schmidt, M. (2007): SiDiff: generische, auf Ähnlichkeiten basierende Berechnung von Modelldifferenzen – Positionspapier: Workshop „Vergleich und Versionierung von UML-Modellen“, SE2007, Hamburg, Deutschland.

[Schm26] Schmidt, F. (1926): Die Anpassung der Betriebe an die Wirtschaftslage. In: Zeitschrift für Betriebswirtschaft 3, S. 85–106.

[Schm28] Schmalenbach, E. (1928): Die Betriebswirtschaftslehre an der Schwelle der neuen Wirtschaftsverfassung. In: Zeitschrift für handelswissenschaftliche Forschung (22), S. 241–251.

[Schm97] Schmitt, H.-J. (1993): Client-Server Architekturen - Architekturmodelle für eine neue informationstechnische Infrastruktur. Lang, Frankfurt a. M..

[Schr11] Schreier, S. (2011): Modeling RESTful applications. In: Pautasso, C. (Hrsg.): Proceedings of the Second International Workshop on RESTful Design (WS-REST 2011). ACM, New York, NY, S. 15–21.

[Schu96] Schulte, R. W. (1996): "Service Oriented" Architectures, Part 2. In: Gartner Research.

[ScNa96] Schulte, R. W.; Natis, Y. V. (1996): "Service Oriented" Architectures, Part 1. In: Gartner Research.

[SCS99] Staehle, W. H.; Conrad, P.; Sydow, J. (1999): Management – Eine verhaltenswissenschaftliche Perspektive. 8. Aufl. (überarbeitet von Peter Conrad), Vahlen, München.

[ScWi04] Schenk, M.; Wirth, S. (2004): Fabrikplanung und Fabrikbetrieb – Methoden für die wandlungsfähige und vernetzte Fabrik. Springer, Berlin.

[SeBi13] Setzer, T.; Bichler, M. (2013): Web-Service-Technologien. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[SeKü04] Sendall, S.; Küster, J. (2004): Taming Model Round-Trip Engineering. In: Proceedings of the Workshop Best Practices for Model-Driven Software Development.

[SGL07] Sheth, A.; Gomadam, K.; Lathem, J. (2007): SA-REST: Semantically Interoperable and Easier- to-Use Services and Mashups. In: IEEE Internet Computing 11 (November/ December), S. 84–87.

[ShMo98] Shewchuk, J. P.; Moodie, C. L. (1998): Definition and Classification of Manufacturing Flexibility Types and Measures. In: The International Journal of Flexible Manufacturing Systems 10, S. 325–349.

[Sied07] Siedersleben, J. (2007): SOA revisited: Komponentenorientierung bei Systemlandschaften. In: Wirtschaftsinformatik 49 (Sonderheft), S. 110–117.

[Sinz13a] Sinz, E. J. (2013): Gestaltung von Informationssystem-Architekturen – Methoden, Modelle, Werkzeuge. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[Sinz13b] Sinz, E. J. (2013): Softwarearchitektur. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[Sinz13c] Sinz, E. J. (2013): Modellierung betrieblicher Informationssysteme. Vorlesung, Wintersemester 2013/2014.

[Sinz14] Sinz, E. J. (2014): Datenmanagementsysteme. Vorlesung, Sommersemester 2014.

[Sinz87] Sinz, E. J. (1987): Datenmodellierung betrieblicher Probleme und ihre Unterstützung durch ein wissensbasiertes Entwicklungssystem. Habilitationsschrift, Regensburg.

[Sinz88b] Sinz, E. J. (1988): Das Strukturierte Entity-Relationship-Modell (SER-Modell). In: Angewandte Informatik 30 (5), S. 191–202.

[Sinz95] Sinz, E. J. (1995): Ein Architekturrahmen für die Modellierung betrieblicher Informationssysteme. In: Bamberger Beiträge zur Wirtschaftsinformatik (32).

[Sinz96] Sinz, E. J. (1996): Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme. Entwicklung, aktueller Stand und Trends. In: Heilmann, H.; Heinrich, L. J.; Roithmayr, F. (Hrsg.): Information Engineering. Wirtschaftsinformatik im Schnittpunkt von Wirtschafts-, Sozial- und Ingenieurwissenschaften. Oldenbourg, München, Wien, S. 123–143.

[Sinz97] Sinz, E. J. (1997): Architektur betrieblicher Informationssysteme. In: Bamberger Beiträge zur Wirtschaftsinformatik (40).

[Sinz98] Sinz, E. J. (1998): Modellierung betrieblicher Informationssysteme: Gegenstand, Anforderungen und Lösungsansätze. In: Pohl, K.; Schürr, A.; Vossen, G. (Hrsg.): Proceedings des GI-Workshops Modellierung ’98, 11.–13. März, 1998, Münster. CEUR-WS.org, S. 27–28.

[SITA13] SITA (2013): The Airline IT Trends Survey. URL: http://www.sita.aero/surveys-reports/industry-surveys-reports/airline-it-trends-survey-2013
(Abruf: 07.05.2013).

[SKL14] Sporny, M.; Kellogg, G.; Lanthaler, M. (2014): JSON-LD 1.0. A JSON-based Serialization for Linked Data – W3C Recommendation 16 January 2014. URL: http://www.w3.org/TR/2014/REC-json-ld-20140116/ (Abruf: 07.05.2014).

[SLS+08] Siikarla, M.; Laitkorpi, M.; Selonen, P.; Systä, T. (2008): Transformations Have to be Developed ReST Assured. In: Vallecillo, A.; Gray, J.; Pierantonio, A. (Hrsg.): Theory and Practice of Model Transformations. Springer, Berlin, Heidelberg, S. 1–15.

[Snel04] Snell, J. (2004): Resource-oriented vs. activity-oriented web services – A quick look at the relationship of rest-style and soap-style web services.
URL: http://www.ibm.com/developerworks/webservices/library/ws-restvsoap/
(Abruf: 18.07.2011 (am 28.04.2014 nicht mehr online verfügbar)).

[Somm11] Sommerville, I. (2011): Software engineering. 9. Aufl., Pearson, Boston.

[Somm12] Sommerville, I. (2012): Software Engineering. 9. Aufl. (deutsche Ausgabe), Pearson, München.

[Stac73] Stachowiak, H. (1973): Allgemeine Modelltheorie. Springer, Wien.

[StAl11] Steiner, T.; Algermissen, J. (2011): Fulfilling the Hypermedia Constraint Via HTTP OPTIONS, the HTTP Vocabulary in RDF, and Link Headers. In: Pautasso, C. (Hrsg.): Proceedings of the Second International Workshop on RESTful Design (WS-REST 2011). ACM, New York, NY, S. 11–14.

[Star11] Starke, G. (2011): Effektive Software-Architekturen – Ein praktischer Leitfaden. 5. Aufl., Hanser, München.

[Stei93] Steinmüller, W. (1993): Informationstechnologie und Gesellschaft – Einführung in die angewandte Informatik. Wissenschaftliche Buchgesellschaft, Darmstadt.

[Stra13a] Strahringer, S. (2013): Konzeptuelle Modellierung von IS. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[Stra13b] Strahringer, S. (2013): Metamodell. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[Stra13c] Strahringer, S. (2013): Modell. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[StTi07a] Starke, G.; Tilkov, S. (Hrsg.): SOA-Expertenwissen – Methoden, Konzepte und Praxis serviceorientierter Architekturen. dpunkt, Heidelberg (2007).

[StTi07b] Starke, G.; Tilkov, S. (2007): Das Einmaleins der serviceorientierten Architekturen. In: Starke, G.; Tilkov, S. (Hrsg.): SOA-Expertenwissen – Methoden, Konzepte und Praxis serviceorientierter Architekturen. dpunkt, Heidelberg, S. 9–36.

[SVE+07] Stahl, T.; Völter, M.; Efftinge, S.; Haase, A. (2007): Modellgetriebene Softwareentwicklung – Techniken, Engineering, Management. 2. Aufl., dpunkt, Heidelberg.

[SzKa97] Sztipanovits, J.; Karsai, G. (1997): Model-integrated computing. In: Computer 30 (4), S. 110–111.

[TaVa13] Tavares, N. A. C.; Vale, S. (2013): A Model Driven Approach for the Development of Semantic RESTful Web Services. In: Weippl, E.; Indrawan-Santiago, M.; Steinbauer, M.; Kotsis, G.; Khalil, I. (Hrsg.): Proceedings of International Conference on Information Integration and Web-based Applications & Services (IIWAS '13), S. 290–299.

[TBW+07] Treude, C.; Berlik, S.; Wenzel, S.; Kelter, U. (2007): Difference computation of large models. In: Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT Symposium on the foundations of software engineering. ACM, New York, NY, S. 295–304.

[TeHi01] Terveen, L.; Hill, W. (2001): Human-Computer Collaboration in Recommender Systems. In: Carroll, J. (Hrsg.): Human Computer Interaction in the New Millenium, New York, S. 487-509.

[TeSi12] Teusch, A.; Sinz, E. J. (2012): Konzeptuelle Modellierung partieller SOA. In: Mattfeld, D. C.; Robra-Bissantz, S. (Hrsg.): Multikonferenz Wirtschaftsinformatik 2012. Tagungsband der MKWI 2012. Gito, Berlin, S. 1637–1648.

[Thel03] Thelin, J. (2003): A Comparison of Service-oriented, Resource-oriented, and Object-oriented Architecture Styles – Presentation at Web Services for the Integrated Enterprise. In: OMG's 2nd Workshop On Web Services Modeling,

Architectures, Infrastructures And Standards. URL: http://research.microsoft.com/pubs/117710/3-arch-styles.pdf (Abruf: 29.04.2014).

[Thom05] Thomas, O. (2005): Das Modellverständnis in der Wirtschaftsinformatik – Historie, Literaturanalyse und Begriffsexplikation. In: Veröffentlichungen des Instituts für Wirtschaftsinformatik (184).

[Tilk07] Tilkov, S. (2007): REST - eine Einführung. In: Starke, G.; Tilkov, S. (Hrsg.): SOA-Expertenwissen. Methoden, Konzepte und Praxis serviceorientierter Architekturen. dpunkt, Heidelberg, S. 389–404.

[Tilk11] Tilkov, S. (2011): REST und HTTP – Einsatz der Architektur des Web für Integrationsszenarien. 2. Aufl., dpunkt, Heidelberg, Neckar.

[Tilk14] Tilkov, S. (2014): A Brief Introduction to REST. In: InfoQ.com (Hrsg.): InfoQ eMag: REST, S. 4–9. URL: http://www.infoq.com/minibooks/emag-rest (Abruf: 28.04. 2014).

[TLD+07] Thomas, O.; Leyking, K.; Dreifus, F.; Fellmann, M.; Loos, P. (2007): Service-orientierte Architekturen: Gestaltung, Konfiguration und Ausführung von Geschäftsprozessen. Veröffentlichungen des Instituts für Wirtschaftsinformatik, Saarbrücken.

[Toul06] Toulmé, A. (2006): Presentation of EMF Compare Utility – Position Paper: Eclipse Summit.

[TuLa05] Turner, D. E.; Lankford, W. M. (2005): Information Technology Infrastructure: A Historical Perspective of Flexibility. In: Journal of Information Technology Management 16 (2), S. 37–47.

[Ulle12] Ullenboom, C. (2012): Java 7 - mehr als eine Insel – Das Handbuch zu den Java SE-Bibliotheken. Galileo Press, Bonn.

[Unla13] Unland, R. (2013): Transaktion. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[VaPa09] Valverde, F.; Pastor, O. (2009): Dealing with REST Services in Model-driven Web Engineering Methods. In: Jornadas Cientifico-Técnicas en Servicios Web y SOA (JSWEB), S. 243–250.

[Vino07] Vinoski, S. (2007): REST Eye for the SOA Guy. In: IEEE Internet Computing 11 (1), S. 82–84.

[Vino08a] Vinoski, S. (2008): RESTful Web Services Development Checklist. In: IEEE Internet Computing 12 (6), S. 94–96.

[Vino08b] Vinoski, S. (2008): Serendipitous Reuse. In: IEEE Internet Computing 12 (1), S. 84–87.

[VLA09] Viering, G.; Legner, C.; Ahlemann, F. (2009): The (Lacking) Business Perspective on SOA – Critical Themes in SOA Research. In: Wirtschaftsinformatik Proceedings 2009 (Paper 5). URL: http://aisel.aisnet.org/wi2009/5 (Abruf: 2.10.2014).

[VöBe13] Völter, M.; Benz, S. (2013): DSL engineering – Designing implementing and using domain-specific languages. CreateSpace Independent Publishing Platform.

[Voge09] Vogel, O. (2009): Software-Architektur – Grundlagen - Konzepte - Praxis. 2. Aufl., Spektrum Akademischer Verlag, Heidelberg.

[Voss08] Vossen, G. (2008): Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme. 5. Aufl., Oldenbourg, München.

[VPF+06] Vernadat, F.; Percebois, C.; Farail, P.; Vingerhoeds, R.; Rossignol, A.; Talpin, J.-P.; Chemouil, D. (2006): The TOPCASED project – a Toolkit in OPen source for Critical Aeronautic SystEms Design: International Space System Engineering Conference - Data Systems in Aerospace.

[VSD+12] Verborgh, R.; Steiner, T.; Deursen Van, D.; Coppens, S.; Vallés, J. G.; Walle Van de, Rik (2012): Functional Descriptions as the Bridge between Hypermedia APIs and the Semantic Web. In: Alarcon, R.; Pautasso, C.; Wilde, E. (Hrsg.): Proceedings of the Third International Workshop on RESTful Design (WS-REST 2012), April 17, 2012, Lyon, France, S. 33–40.

[W3C14] W3C (2014): Resource Description Framework (RDF).
URL: http://www.w3.org/RDF/ (Abruf: 25.04.2014).

[WAD+07] Wahli, U.; Ackerman, L.; Di Bari, A.; Hodgkinson, G.; Kesterton, A.; Olson, L.; Portier, B. (2007): Building SOA Solutions Using the Rational SDP. IBM Redbooks.

[Wall96] Wall, F. (1996): Organisation und betriebliche Informationssysteme – Elemente einer Konstruktionstheorie. Gabler, Wiesbaden.

[WiAi13] Winter, R.; Aier, S. (2013): Informationssystem-Architektur. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[WiHe06] Wilde, T.; Hess, T. (2006): Methodenspektrum der Wirtschaftsinformatik – Überblick und Portfoliobildung. In: Arbeitsbericht des Instituts für Wirtschaftsinformatik und Neue Medien (2).

[WiHe07] Wilde, T.; Hess, T. (2007): Forschungsmethoden der Wirtschaftsinformatik – Eine empirische Untersuchung. In: Wirtschaftsinformatik 49 (4), S. 280–287.

[Wiki14] Wikipedia (2014): Uniform Resource Identifier. URL: http://de.wikipedia.org/wiki/Uniform_Resource_Identifier/ (Abruf: 25.04.2014).

[Wild07] Wilde, E. (2007): Putting Things to REST. URL: http://dret.net/netdret/publications#wil07n (Abruf: 25.04.2014).

[Wint13] Wintermeyer, S. (2013): Ruby on Rails 3.2 – Für Ein-, Um- und Quereinsteiger. Addison-Wesley, München.

[WKK+11] Weidmann, M.; Koetter, F.; Kintz, M.; Schleicher, D.; Mietzner, R. (2011): Adaptive Business Process Modeling in the Internet of Services (ABIS). In: Matskin, M. (Hrsg.): Proceedings of the Sixth International Conference on Internet and Web Applications and Services (ICIW 2011), S. 29–34.

[WoBe13] Wolf, M.; Benker, T. (2013): Vom SOM-Geschäftsprozessmodell zur vollständig dokumentenorientierten RESTful SOA – Ein modellbasierter Ansatz. In: Wirtschaftsinformatik Proceedings 2013, Paper 77.
URL: http://aisel.aisnet.org/wi2013/77 (Abruf: 28.09.2014).

[Wolf12] Wolf, M. (2012): Modellbasierte Spezifikation von RESTful SOA auf Basis von Geschäftsprozessmodellen. In: Mattfeld, D. C.; Robra-Bissantz, S. (Hrsg.): Multikonferenz Wirtschaftsinformatik 2012. Tagungsband der MKWI 2012. Gito, Berlin, S. 1649–1660.

[Wolf89] Wolf, J. (1989): Investitionsplanung zur Flexibilisierung der Produktion. Deutscher Universitätsverlag, Wiesbaden.

[Work12] Workflow Management Coalition (WfMC) (2012): XML Process Definition Language (XPDL) – Version 2.2 - WfMC Standard.

URL: http://www.xpdl.org/standards/xpdl-2.2/XPDL%202.2%20%282012-08-30%29.pdf (Abruf: 29.08.2013).

[WöSc13] Wörndl, W.; Schlichter, J. (2013): Empfehlungssysteme. In: Kurbel, K.; Becker, J.; Gronau, N.; Sinz, E. J.; Suhl, L. (Hrsg.): Enzyklopädie der Wirtschaftsinformatik. Online-Lexikon (Stand 14.10.2013). 7. Aufl., De Gruyter Oldenbourg Wissenschaftsverlag, München.

[WPR10] Webber, J.; Parastatidis, S.; Robinson, I. (2010): REST in Practice – Hypermedia and systems architecture. O'Reilly, Beijing.

[WPR14] Webber, J.; Parastatidis, S.; Robinson, I. (2014): How to GET a Cup of Coffee. In: InfoQ.com (Hrsg.): InfoQ eMag: REST, S. 15–28.
URL: http://www.infoq.com/minibooks/emag-rest (Abruf: 15.04.2014).

[WRR07] Weber, B.; Rinderle, S.; Reichert, M. (2007): Change Patterns and Change Support Features in Process-Aware Information Systems. In: Krogstie, J.; Opdahl, A.L.; Sindre, G. (Hrsg.): CAiSE 2007, LNCS 4495, S. 574–588.

[WSL+11a] Wagner, D.; Suchan, C.; Leunig, B.; Frank, J. (2011): Methode zur Analyse der Flexibilität von IS. In: Sinz, E. J.; Bartmann, D.; Bodendorf, F.; Ferstl, O. K. (Hrsg.): Dienstorientierte IT-Systeme für hochflexible Geschäftsprozesse. University of Bamberg Press, Bamberg, S. 79–106.

[WSL+11b] Wagner, D.; Suchan, C.; Leunig, B.; Frank, J. (2011): Towards the Analysis of Information Systems Flexibility: Proposition of a Method. In: Bernstein, A. (Hrsg.): Proceedings of the 10th International Conference on Wirtschaftsinformatik (WI 2011). February 16–18, 2011, Zurich, Switzerland, S. 808–817.

[WSR09] Weber, B.; Sadiq, S.; Reichert, M. (2009): Beyond Rigidity - Dynamic Process Lifecycle Support – A Survey on Dynamic Changes in Process-aware Information Systems. In: Computer Science - Research and Development 23 (2), S. 47–65.

[XiSt05] Xing, Z.; Stroulia, E. (2005): UMLDiff: An Algorithm for Object-Oriented Design Differencing. In: Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering (ASE 2005). ACM, New York, N.Y, S. 54–65.

[XZL+08a] Xu, X.; Zhu, L.; Liu, Y.; Staples, M. (2008a): Resource-Oriented Architecture for Business Processes: Proceedings of the 15th Asia-Pacific Software Engineering Conference. IEEE, S. 395–402.

[XZL+08b] Xu, X.; Zhu, L.; Liu, Y.; Staples, M. (2008b): Resource-oriented business process modeling for ultra-large-scale systems. In: Sullivan, K.; Kazman, R. (Hrsg.): Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems (ULSSIS '08), Leipzig, Germany, S. 65–68.

[YaPa04] Yang, J.; Papazoglou, M. (2004): Service components for managing the life-cycle of service compositions. In: Journal of Information Systems 29 (2), S. 97–125.

[ZhDo09] Zhao, H.; Doshi, P. (2009): Towards Automated RESTful Web Service Composition. In: Damiani, E. (Hrsg.): Proceedings of the 2009 IEEE Internat'l Conference on Web Services (ICWS 2009). IEEE Computer Society; IEEE, Washington, DC, S. 189–196.