Fortschritt-Berichte VDI

VDI

Reihe 10

Informatik/
Kommunikation

Dipl.-Ing. Stephan Hensel
Dresden

Nr. 873

Semantische Revisionskontrolle für die Evolution von Informations- und Datenmodellen

Berichte aus der Professur für **Prozessleittechnik** und der Arbeitsgruppe **Systemverfahrenstechnik** der TU Dresden, Prof. Dr.-Ing. habil. Leon Urbas (Hrsg.)

PCS PSE

TECHNISCHE UNIVERSITÄT DRESDEN

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik

Institut für Automatisierungstechnik

Semantische Revisionskontrolle für die Evolution von Informations- und Datenmodellen

Semantic revision control for the evolution of information and data models

Dipl.-Ing. Stephan Hensel

Der Fakultät Elektrotechnik und Informationstechnik
der Technischen Universität Dresden

zur Erlangung des akademischen Grades eines

Doktoringenieurs
(Dr.-Ing.)

genehmigte
Dissertation

Vorsitzender:	Prof. Dr.-Ing. Dr. h.c. Frank H. P. Fitzek
Gutachter:	Prof. Dr.-Ing. habil. Leon Urbas
Gutachter:	Univ.-Prof. Dr.-Ing. Alexander Fay
Gutachter:	Prof. Dr.-Ing. Christian Diedrich

Tag der Einreichung:	16.05.2019
Tag der Verteidigung:	20.11.2020

Fortschritt-Berichte VDI

Reihe 10

Informatik/
Kommunikation

Dipl.-Ing. Stephan Hensel,
Dresden

Nr. 873

Semantische Revisionskontrolle für die Evolution von Informations- und Datenmodellen

Berichte aus der Professur für **Prozessleittechnik** und der Arbeitsgruppe **Systemverfahrenstechnik** der TU Dresden, Prof. Dr.-Ing. habil. Leon Urbas (Hrsg.)

PCS PSE

TECHNISCHE UNIVERSITÄT DRESDEN

Hensel, Stephan
Semantische Revisionskontrolle für die Evolution von Informations- und Datenmodellen
Fortschr.-Ber. VDI Reihe 10 Nr. 873. Düsseldorf: VDI Verlag 2021.
188 Seiten, 64 Bilder, 5 Tabellen.
ISBN 978-3-18-387310-4, ISSN 0178-9627,
€ 67,00/VDI-Mitgliederpreis € 60,30.

Keywords: Semantik – Revisionskontrolle – Evolution – Informationsmodelle – Datenmodelle – Linked Data – R43ples – Co-Simulation – Modularisierung – Module Type Package

Im Rahmen dieser Dissertation wurde ein Revision Management System zur durchgängigen Unterstützung der Evolution von Informations- und Datenmodellen entwickelt, das Revisionsverwaltungs- und Evolutionsmechanismen integriert. Besonderheit ist hierbei die technologieunabhängige mathematische und semantische Beschreibung, die eine Überführung des Konzepts in unterschiedliche Technologien ermöglicht. Beispielhaft wurde das Konzept für das Semantic Web als Weiterentwicklung des Open-Source-Projektes R43ples umgesetzt.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter www.dnb.de abrufbar.

Bibliographic information published by the Deutsche Bibliothek
(German National Library)
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie (German National Bibliography); detailed bibliographic data is available via Internet at www.dnb.de.

Als Manuskript gedruckt. Printed in Germany.
ISSN 0178-9627
ISBN 978-3-18-387310-4

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich in der Zeit als wissenschaftlicher Mitarbeiter und Doktorand an der Professur für Prozessleittechnik und Arbeitsgruppe Systemverfahrenstechnik fachlich und persönlich unterstützt haben.

Ein besonderer Dank gilt Herrn Prof. Dr.-Ing. habil. Leon Urbas, meinem Doktorvater, für die Betreuung dieser Arbeit und die Bereitstellung der notwendigen Infrastruktur für die Umsetzung der Arbeit in einem sehr spannenden Themenkomplex. Durch die zahlreichen Projekte hat er mir viele Einblicke in die industrielle Praxis ermöglicht, wodurch ich auch wichtige Kontakte für meine weitere berufliche Zukunft knüpfen konnte. Seine kritischen Nachfragen und die gemeinsamen wissenschaftlichen Diskussionen haben wesentlich zur Ideenfindung für diese Dissertation beigetragen.

Ich danke Herrn Univ.-Prof. Dr.-Ing. Alexander Fay für die sehr gute Zusammenarbeit und die Übernahme der Zweitbegutachtung.

Meiner Familie möchte ich dafür danken, dass sie mich während all der Zeit so herzlich unterstützt hat und immer verständnisvoll war, dass die Erstellung der Dissertation viel Zeit in Anspruch genommen hat.

Weiterhin danke ich den zahlreichen Korrekturleserinnen und Korrekturlesern, die sehr viel Zeit in die Überprüfung von Rechtschreibung, Kommasetzung und vielen sprachlichen Kleinigkeiten investiert haben.

Außerdem möchte ich mich bei meinen Kollegen für die gute Zeit am Lehrstuhl, die vielen gemeinsamen Dienstreisen und den intensiven wissenschaftlichen Diskurs bedanken.

Schließlich danke ich Dr. Jürgen Hambrecht und Eggert Voscherau, den Gründern der HaVo-Stiftung, für die finanzielle Unterstützung im Rahmen eines HaVo-Stipendiums.

Inhaltsverzeichnis

Abkürzungs- und Symbolverzeichnis

Abkürzungen

ACID	Atomicity, Consistency, Isolation, Durability
AERO	Aggregation and Evolution Rules Ontology
AutomationML	Automation Markup Language
BASE	Basically Available, Soft state, Eventual consistency
BPMN	Business Process Model and Notation
CAP	Consistency, Availability, Partition Tolerance
CHAO	Change and Annotation Ontology
CIF	Continuous Integration Framework
CPS	Cyber Physical Systems
CVS	Concurrent Versions System
DIMA	Dezentrale Intelligenz für modulare Anlagen
EMF	Eclipse Modeling Framework
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
GLD	Goverment Linked Data
IP	Internetprotokoll
KAON	Karlsruhe Ontology and Semantic Web framework
LDAP	Lightweight Directory Access Protocol
LED	Linked Enterprise Data
LOD	Linked Open Data
MDE	Model-driven Engineering
MMO	Merge Management Ontology

MOF	Meta Object Facility
MTP	Module Type Package
OPC UA	Open Platform Communications Unified Architecture
OWL	Web Ontology Language
PEA	Process Equipment Assembly
PFE	Prozessführungsebene
POL	Process Orchestration Layer
PROV-O	PROV Ontology
QUDT	Quantities, Units, Dimensions, and Data Types
RCS	Revision Control System
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RMO	Revision Management Ontology
RMS	Revision Management System
SCCS	Source Code Control System
SHACL	Shapes Constraint Language
SPARQL	SPARQL Protocol And RDF Query Language
SPIN	SPARQL Inferencing Notation
SVN	Apache Subversion
TGG	Tripel-Graph-Grammatik
UML	Unified Modeling Language
URI	Uniform Resource Identifier
USB	Universal Serial Bus
VIBN	Virtuelle Inbetriebnahme
XML	Extensible Markup Language

Symbole und Funktionen

Allgemein

$\mathbb{N}^+$	Menge der natürlichen Zahlen größer Null
$\mathbb{N}_0^+$	Menge der natürlichen Zahlen größer gleich Null
$\lvert\mathcal{X}\rvert$	Mächtigkeit der Menge $\mathcal{X}$
$\mathcal{P}(M)$	Potenzmenge einer Menge M

Compound Graphs

$\tilde{G}$	Compound Graph
$\tilde{G}'$	Einfach gerichteter Graph innerhalb des Compound Graphs
$\tilde{T}'$	Baum innerhalb des Compond Graphs
$\hat{G}$	Einfach gerichteter Graph
$\hat{T}$	Baum
$\hat{V}$	Menge an Knoten
$\hat{E}$	Menge an Kanten
$\hat{v}$,$\hat{w}$	Knoten aus $\hat{V}$
$\boldsymbol{pred}_{\hat{G}}(\hat{v})$	Funktion zur Ermittlung von Vorgängern in $\hat{G}$
$\boldsymbol{succ}_{\hat{G}}(\hat{v})$	Funktion zur Ermittlung von Nachfolgern in $\hat{G}$
$\tilde{B}$	Menge der Basisknoten (Blättern von $\tilde{T}'$)
$\tilde{S}$	Menge der Subgraphen (innere Knoten von $\tilde{T}'$)
$\tilde{V}$	Menge der Knoten als Vereinigung von $\tilde{B}$ und $\tilde{S}$
$\tilde{b}$,$\tilde{s}$,$\tilde{v}$,$\tilde{w}$	Knoten aus $\tilde{V}$
$\tilde{E}_{\tilde{G}'}$	Adjazenzkanten
$\tilde{E}_{\tilde{T}'}$	Inklusionskanten
$\tilde{n}_v$	Eindeutiger Identifikator des Knotens
$\tilde{n}_z$	Eindeutiger Identifikator zur Zuordnung der Semantik zum Knoten

Änderungsmanagement

$\mathcal{S}$	Menge aller Statements
$\mathcal{G}$	Revisionsgraph
$\mathcal{R}$	Menge aller möglichen Revisionen
$\mathcal{C}$	Menge aller möglichen Änderungen zwischen zwei Revisionen (ChangeSets)
$\mathcal{B}$	Menge aller möglichen Entwicklungszweige (Branches)
$\mathcal{T}$	Menge aller möglichen Tags
R_g	Menge der Revisionen innerhalb von $\mathcal{G}$
C_g	Menge der Änderungen zwischen zwei Revisionen (ChangeSets) innerhalb von $\mathcal{G}$
B_g	Menge der Entwicklungszweige (Branches) innerhalb von $\mathcal{G}$
T_g	Menge der Tags innerhalb von $\mathcal{G}$
n_g	Eindeutiger Identifikator eines Revisionsgraphen in der Menge der Revisionsgraphen
C^+	Menge der hinzugefügten Elemente
C^-	Menge der gelöschten Elemente
r_l	Blattrevision des Entwicklungszweiges
r_x,r_y,r_t	Revisionen aus $\mathcal{R}$
R_b	Menge der Revisionen eines Entwicklungszweiges
n_b	Eindeutiger Identifikator eines Branches im Revisionsgraphen
n_t	Eindeutiger Identifikator eines Tags im Revisionsgraphen
Υ_l	Vollständiger Revisionsinhalt des Blattes des Entwicklungszweiges
Υ_t	Vollständiger Revisionsinhalt eines Tags
$\boldsymbol{pred}_{\mathcal{G}}(r_y)$	Funktion zur Ermittlung von Vorgängerrevisionen in $\mathcal{G}$
$\boldsymbol{succ}_{\mathcal{G}}(r_x)$	Funktion zur Ermittlung von Nachfolgerrevisionen in $\mathcal{G}$
$\boldsymbol{path}_{\mathcal{G}}(r_x,r_y)$	Funktion zur Ermittlung eines Revisionspfades in $\mathcal{G}$
$\boldsymbol{getContent}_{\mathcal{G}}(r_x)$	Funktion zur Wiederherstellung des vollständigen Revisionsinhalts

$\boldsymbol{strip}(\Upsilon_x, C^+, C^-)$	Funktion zur Berechnung von minimalen ChangeSets
Γ	Menge der Revisionsgraphen
$\boldsymbol{create}_\Gamma$	Funktion zur Erstellung eines neuen Revisionsgraphen
$\boldsymbol{drop}_\Gamma$	Funktion zur Löschung eines bestehenden Revisionsgraphen
$\boldsymbol{branch}_\mathcal{G}$	Funktion zur Erstellung eines neuen Entwicklungszweiges
$\boldsymbol{tag}_\mathcal{G}$	Funktion zur Erstellung eines neuen Tags
Υ_x	Vollständiger Revisionsinhalt einer spezifizierten Revision
r^*	Neu erstellte Revision
$\boldsymbol{commit}_\mathcal{G}$	Funktion zur Erstellung eines neuen Commits
$\boldsymbol{revert}_\mathcal{G}$	Funktion, um einen vorher erstellten Commit rückgängig zu machen

Aggregation von High-Level-Changes

$\boldsymbol{hlcAgg}_\mathcal{G}$	Funktion zur Aggregation von atomaren Änderungen zu High-Level-Changes
$\boldsymbol{\Phi}_\mathcal{G}$	Funktion zur Berechnung von High-Level-Changes zwischen zwei Revisionen
n_z	Eindeutiger Identifikator des High-Level-Changes
C_r^+	Menge der nicht zuzuordnenden hinzugefügten Elemente
C_r^-	Menge der nicht zuzuordnenden gelöschten Elemente

Zusammenführung divergierter Entwicklungszweige

$\boldsymbol{merge}_\mathcal{G}$	Funktion zur Zusammenführung von divergierten Entwicklungszweigen mittels eines 3-Wege-Merges
b_s	Quellentwicklungszweig
b_t	Zielentwicklungszweig
$\boldsymbol{\Psi}_\boldsymbol{G}$	Funktion zur Berechnung der ChangeSets in Bezug auf die jeweiligen Entwicklungszweige
$\boldsymbol{pick}_\mathcal{G}$	Funktion zur Wiederverwendung von bereits durchgeführten Änderungen in Bezug auf eine Revision
r_p	Revision, die wiederverwendet werden soll

$\vec{R_p}$	Geordnete Liste von Revisionen
$\boldsymbol{pick_{\mathcal{G}}}\left(\vec{R_p}, n_b\right)$	Funktion zur Wiederverwendung einer Liste an bestehenden Revisonen
$\boldsymbol{fastForward_{\mathcal{G}}}$	Funktion zur Zusammenführung von divergierten Entwicklungszweigen, wobei Revisionshistorie geglättet wird und nicht notwendige 3-Wege-Merges vermieden werden
$\mathcal{K}$	Menge der möglichen Status von atomaren Änderungen
$\boldsymbol{getCommonAncestor_{\mathcal{G}}}\left(n_{b_s}, n_{b_t}\right)$	Funktion zur Berechnung der gemeinsamen Vorgängerrevision von zwei Entwicklungszweigen
r_c	Gemeinsame Vorgängerrevision von zwei Entwicklungszweigen
C_{path_s}	Pfad von gemeinsamer Vorgängerrevision zum Blatt des Quellentwicklungszweiges
C_{path_t}	Pfad von gemeinsamer Vorgängerrevision zum Blatt des Zielentwicklungszweiges
Ω_{Start}	Startmenge für das Nachvollziehen der Änderungen
$\boldsymbol{add}(\Omega, s)$	Funktion zur Aktualisierung des Status eines Statements, wenn dieses hinzugefügt wird
$\boldsymbol{del}(\Omega, s)$	Funktion zur Aktualisierung des Status eines Statements, wenn dieses gelöscht wird
s	Einzelnes Statement
$\boldsymbol{F_{C_n}}(\Omega)$	Funktion zu Anwendung der Aktualisierungen von einer Revision
Ω_{End}	Endmenge mit allen Status nach dem Nachvollziehen der Änderungen
$\mathcal{Q}$	Menge an Definitionen zur automatisierten Konfliktbehebung
k_s	Status Quellentwicklungszweig
k_t	Status Zielentwicklungszweig
q	Boolesche Größe zur Spezifikation, ob es sich um einen Konflikt handelt oder nicht
$\mathcal{D}_s$	Menge der Status auf dem Quellentwicklungszweig ohne gleiche Status des Zielentwicklungszweiges
$\mathcal{D}_t$	Menge der Status auf dem Zielentwicklungszweig ohne gleiche Status des Quellentwicklungszweiges

$\tilde{\mathcal{D}}_s$ Menge der Status auf dem Quellentwicklungszweig mit gleicher Kardinalität, wie Zielentwicklungszweig

$\tilde{\mathcal{D}}_t$ Menge der Status auf dem Zielentwicklungszweig mit gleicher Kardinalität, wie Quellentwicklungszweig

$\mathcal{D}_{Diff}$ Zusammenführung der Mengen $\tilde{\mathcal{D}}_s$ und $\tilde{\mathcal{D}}_t$ zur Beschreibung der Unterschiede

$\mathcal{D}_Q$ Ergebnismenge der Konflikterkennung und -behebung mit entsprechenden Status für eine automatisierte Konfliktbehebung

$\boldsymbol{hlcPathAgg}_{\mathcal{G}}\,(C_{path})$ Funktion zur Aggregation von atomaren Änderungen zu High-Level-Changes entlang eines Pfades

$\boldsymbol{intersec}_{\mathcal{G}}\,(C_{path})$ Funktion zur Berechnung von Abhängigkeiten von High-Level-Changes

Evolutionsmechanismen

$\mathcal{G}_e$ Revisionsgraph zur semantischen Beschreibung von durchgeführten Co-Evolutionen

$h_{\mathcal{G}_s}$ Ergebnis der High-Level-Change-Aggregation der zu co-evolvierenden Änderungen

$\boldsymbol{calcDep}_{\Gamma}\,(\mathcal{G}_s,r_{x_s},r_{y_s},h_{\mathcal{G}_s})$ Funktion zur Berechnung von Abhängigkeiten

$\mathcal{G}_s$ Quellrevisionsgraph

$\mathcal{G}_t$ Zielrevisionsgraph

$\boldsymbol{coevolve}_{\Gamma}\,(h_{\mathcal{G}_s},\mathcal{G}_t,n_{b_t})$ Funktion zur Erstellung von Co-Evolutionscommits auf abhängige Entwicklungszweige

$\mathcal{E}\,(h_{\mathcal{G}_s},\mathcal{G}_t,n_{b_t})$ Funktion zur Berechnung der Hinzufügungen und Löschen für die Co-Evolution

$\boldsymbol{coevolveAll}_{\Gamma}\,(\mathcal{G}_s,r_{x_s},r_{y_s})$ Funktion zur Co-Evolution von allen abhängigen Revisionsgraphen und Entwicklungszweigen

$\mathcal{Z}$ Menge zur temporären Speicherung der durchgeführten Änderungen

$\Xi(\mathcal{Z})$ Funktion zur Überführung der durchgeführten Änderungen in Hinzufügungen und Löschungen zur semantischen Beschreibung

Implementierung

$\mathcal{L}$	Menge aller Literale
$\mathcal{U}$	Menge aller Uniform Resource Identifiers

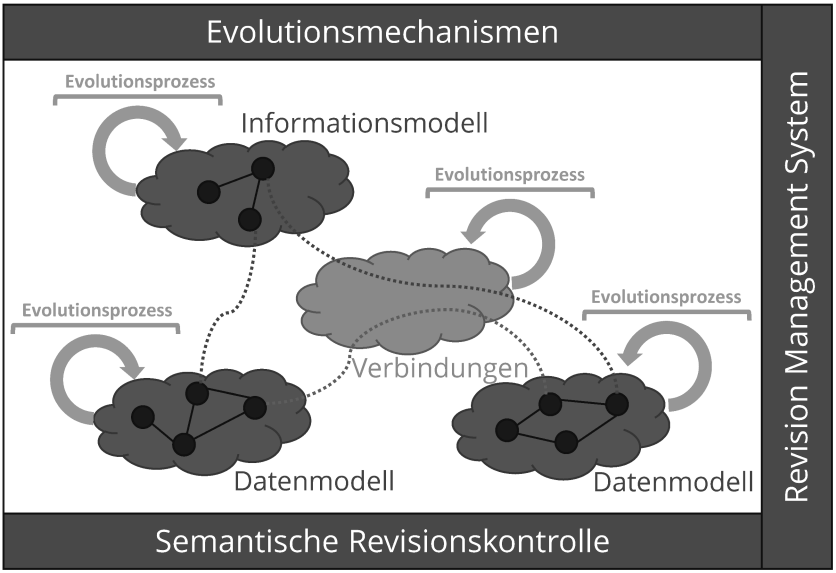
Verifikation

$\tilde{I}$	Menge der Eingangsports
$\tilde{O}$	Menge der Ausgangsports
$d^-_{\tilde{G}'}(\tilde{v})$	Eingangsgrad eines Knotens $\tilde{v}$ in $\tilde{G}'$
$d^+_{\tilde{G}'}(\tilde{v})$	Ausgangsgrad eines Knotens $\tilde{v}$ in $\tilde{G}'$
$\tilde{E}_{\tilde{G}'_{connection}}$	Menge der Verbindungen über Komponenten hinweg
$\tilde{E}_{\tilde{G}'_{dependency}}$	Menge der internen Abhängigkeiten innerhalb einer Komponente

Kurzfassung

Semantische Revisionskontrolle für die Evolution von Informations- und Datenmodellen

Stärker verteilte Systeme in der Planung und Produktion verbessern die Agilität und Wartbarkeit von Einzelkomponenten, wobei gleichzeitig jedoch deren Vernetzung untereinander steigt. Das stellt wiederum neue Anforderungen an die semantische Beschreibung der Komponenten und deren Verbindungen, wofür Informations- und Datenmodelle unabdingbar sind. Der Lebenszyklus dieser Modelle ist dabei von Änderungen geprägt, mit denen umgegangen werden muss. Heutige Revisionsverwaltungssysteme, die die industriell geforderte Nachvollziehbarkeit bereitstellen könnten, sind allerdings nicht auf die speziellen Anforderungen der Informations- und Datenmodelle zugeschnitten, wodurch Möglichkeiten einer konsistenten Evolution verringert werden.

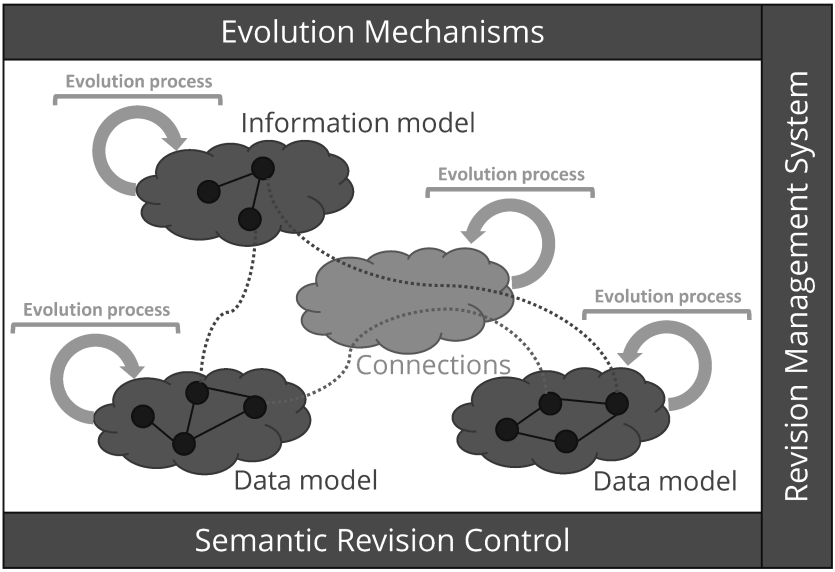


Im Rahmen dieser Dissertation wurde ein Revision Management System zur durchgängigen Unterstützung der Evolution von Informations- und Datenmodellen entwickelt, das Revisionsverwaltungs- und Evolutionsmechanismen integriert. Besonderheit ist hierbei die technologieunabhängige mathematische und semantische Beschreibung, die eine Überführung des Konzepts in unterschiedliche Technologien ermöglicht. Beispielhaft wurde das Konzept für das Semantic Web als Weiterentwicklung des Open-Source-Projektes R43ples umgesetzt.

Abstract

Semantic revision control for the evolution of information and data models

The increased distribution of systems in planning and production leads to improved agility and maintainability of individual components, whereas concurrently their cross-linking increases. This causes new requirements for the semantic description of components and links for which information and data models are indispensable. The life cycle of those models is characterized by changes that must be dealt with. However, today's revision control systems would provide the required industrial traceability but are not enough for the specific requirements of information and data models. As a result, possibilities for a consistent evolution are reduced.



Within this thesis a revision management system was developed, integrating revision control and evolution mechanisms to support the evolution of information and data models. The key is the technology-independent mathematical and sematic description allowing the application of the concept within different technologies. Exemplarily the concept was implemented for the Semantic Web as an extension of the open source project R43ples.

1 Einleitung

1.1 Motivation

Begriffe wie Industrie 4.0, Cyber Physical Systems (CPS), Digital Twin [NFM17], Digital Companion [Sie18], Modularisierung [NAM13] oder Smart Equipment [SUW18] prägen die Industrie von heute. Grund hierfür sind neue Herausforderungen, die unter anderem aus der Globalisierung der Märkte sowie einer Diversifizierung der Produktpalette resultieren [Bie+16]. Mit all diesen Begriffen sind Lösungsmöglichkeiten verbunden, um diesen Herausforderungen begegnen zu können. Jedoch geht mit deren Umsetzung auch eine immer weiter fortschreitende Digitalisierung einher, die zu stärker verteilten Systemen in der Planung und Produktion führt. Funktionen die neu hinzugefügt werden, werden häufig in eigene Systeme ausgelagert, was eine lose Kopplung mit anderen Komponenten ermöglicht. Dies hat den Vorteil, dass die Einzelkomponenten agil weiterentwickelt und wartbar gestaltet werden können. Für die Kommunikation zwischen den Komponenten aber auch zur semantischen Beschreibung der in den jeweiligen Systemen vorgehaltenen Daten sind dabei Informationsmodelle unabdingbar. Diese Informationsmodelle unterliegen, ähnlich wie Produkte, einem Lebenszyklus, der dadurch geprägt ist, dass auf Anforderungsänderungen reagiert werden muss und entsprechende Anpassungen an den Modellen vorgenommen werden müssen. Insbesondere durch die immer kürzer werdenden Produktlebenszyklen entstehen hieraus Änderungswünsche an den Modellen, mit denen umgegangen werden muss. Dabei muss einer Architekturerosion vorgebeugt werden, da diese zu einer Verschlechterung der Struktur führt und weitere Änderungen nur erschwert oder nicht umgesetzt werden können [RB09]. Notwendige Änderungen an den zugehörigen Informationsmodellen werden hierbei nicht abrupt durchgeführt, sondern erfolgen zumeist in kleineren Schritten [Lev+10]. Beispiele für solche kontinuierlichen Evolutionen, die durchaus über mehrere Jahre hinweg gegeben sein können, sind unter anderem Unified Modeling Language (UML) und Business Process Model and Notation (BPMN) [HKB17].

Insbesondere in industriellen Anwendungen spielt jedoch die Dokumentation und die damit verbundene Nachvollziehbarkeit der Änderungen eine wesentliche Rolle, da die Unternehmen gesetzlichen Regularien und damit verbundenen Nachweispflichten unterworfen sind. Ein Beispiel hierfür ist die Prozessindustrie, die sich mit neuen Herausforderungen, wie hoch-volatile Märkte und kürzere Produktlebenszyklen, konfrontiert sieht und mit der Modularisierung verfahrenstechnischer Anlagen auf die Anforderungsänderungen reagiert. Für die Beschreibung modularer Anlagen werden aber wiederum Informationsmodelle benötigt, die die einzelnen Module[1)] und die Verbindungen zwischen diesen beschreiben. Auch hier gilt, dass Veränderungen an Modellen nachvollzogen sowie syntaktische und semantische Korrektheit überprüft werden müssen [Lev+10]. Zur

[1)] Der Begriff Modul wird in dieser Arbeit synonym zu Process Equipment Assembly (PEA) (definiert in [VDI19]) verwendet.

Erreichung dieses Ziels ist eine Integration von Revisionsverwaltungs- und Evolutionsmechanismen, die Änderungen an verknüpfte Modelle propagieren können, unabdingbar [NK04]. Bestehende Ansätze aus dem Bereich der Softwareentwicklung funktionieren hierfür nur bedingt. Dies liegt auf der einen Seite an der zusätzlichen Flexibilisierung aufgrund von Ansätzen wie Linked Enterprise Data (LED) [Gra16], aber auf der anderen Seite auch an der Natur der Modelle, da diese als ein Graph interpretiert werden können [Lev+10]. Bei der Verwendung von etablierten Werkzeugen aus der Softwareentwicklung geht daher jedoch zumeist die Semantik der Änderungen verloren [GHU14]. An dieser Stelle setzt diese Arbeit an, mit dem Ziel, die Semantik der durchgeführten Änderungen zu erhalten, um damit die Grundlage für durchzuführende Evolutionen zu schaffen.

1.2 Zielstellung und erwartete Ergebnisse

Zielstellung dieser Arbeit ist die durchgängige Unterstützung der Evolution von Informations- und Datenmodellen über deren Lebenszyklus hinweg. Hierfür werden Funktionalitäten der Revisionsverwaltung und Evolutionsmechanismen in einem gemeinsamen semantischen Framework integriert, das als Revision Management System (RMS) bezeichnet wird. Ergebnis der Arbeit sind die formalen, technologieunabhängigen Grundlagen für die Umsetzung von Revisionsverwaltungsfunktionalitäten und Evolutionsmechanismen für Informations- und Datenmodelle, die Konzeption eines RMS und eine prototypische Implementierung im Semantic Web.

1.2.1 Kernthese

Der Arbeit wird folgende Kernthese zugrunde gelegt:

> *Ein Revision Management System unterstützt die Evolution von Informations- und Datenmodellen über deren gesamten Lebenszyklus durch die Integration von Revisionskontroll- und Evolutionsmechanismen.*

1.2.2 Einzelthesen

Nachfolgend wird die Kernthese in Forschungsthesen zerlegt:

These 1: Neue Anforderungen an die Agilität von Produktlebenszyklen erfordern Veränderungen im Lebenszyklus der zugrundeliegenden Informationsräume, vor allem im Bereich der Revisionierung und Evolution der Informations- und Datenmodelle.

These 2: Die Anforderungen können durch etablierte Werkzeuge aus der Softwareentwicklung nicht vollständig erfüllt werden.

These 3: Die Integration von Revisionskontrollfunktionalitäten und Evolutionsmechanismen in ein übergeordnetes Revision Management System bietet die Grundlage für die Umsetzung der Anforderungen.

These 4: Die technologieunabhängige Beschreibung des Revision Management Systems erlaubt eine Umsetzung in unterschiedlichen Anwendungsdomänen.

1.3 Einordnung und Abgrenzung der Arbeit

Allgemein ist diese Arbeit im Bereich der Informationsmodellierung angesiedelt. Konkret beschäftigt sie sich mit der Revisionsverwaltung und der Evolution von Informations- und Datenmodellen über den gesamten Lebenszyklus dieser Modelle hinweg. Hieraus entstehen Querbezüge zu verwandten Themengebieten, wie Revisionsverswaltung und Evolution in der Softwareentwicklung [Vog+15b] oder der Schemaevolution im Bereich der Datenbanken [Lev+10]. Ebenso kann auf Vorarbeiten im Bereich des Semantic Webs, wie [Pap+13] und [AM17], zurückgegriffen werden. Aus den aufgeführten Themengebieten können einerseits Anforderungen aber andererseits auch Konzepte extrahiert werden, die auf die speziellen Gegebenheiten von Informations- und Datenmodelle hin angepasst und generalisiert werden. Dabei werden ebenfalls bereits existente Arbeiten wie [Sto04] und [Keh15] aus dem Bereich der Evolution von Informations- und Datenmodelle einbezogen. Eine wesentliche Unterscheidung im Vergleich zu existenten Ansätzen ist die Integration von Revisionskontrollfunktionalitäten und Evolutionsmechanismen.

Insbesondere in integrierten Informationsräumen, wie zum Beispiel in [Gra16] vorgestellt, entsteht eine stärkere Vernetzung der Modelle. Aufgrund der Abhängigkeiten zwischen den unterschiedlichen Modellen müssen durchgeführte Evolutionen an einem Modell an abhängige Modelle coevolviert werden, um stets Konsistenz sicherzustellen. Für die möglichst automatisierte Durchführung dieser Co-Evolutionen, unter Gewährleistung einer durchgängigen Nachvollziehbarkeit der durchgeführten Änderungen, werden innerhalb dieser Arbeit die formalen und technologieunabhängigen Grundlagen für die Umsetzung von Revisionsverwaltungsfunktionalitäten und Evolutionsmechanismen gelegt. Hierfür werden mathematische und semantische Beschreibungen verwendet, um die bereitzustellenden Funktionalitäten, wie zum Beispiel Commits, Zusammenführungen und Co-Evolutionen, zu beschreiben. Diese orientieren sich dabei an Funktionalitäten aus der Entwicklung von Softwaresystemen sowie technologieabhängigen Teillösungen und werden innerhalb dieser Arbeit auf die spezifischen Anforderungen der Informations- und Datenmodelle hin angepasst. Durch die Integration von Revisionsverwaltung und Evolution wird ein RMS geschaffen. Dieses soll über den gesamten Lebenszyklus der Informations- und Datenmodelle hinweg eingesetzt werden und die Evolution und Konsistenzsicherung der enthaltenen Modelle unterstützen.

Bei den betrachteten Abhängigkeiten wird sich in dieser Arbeit auf die bereits vorhandene Vernetzung der Modelle fokussiert. Diese ist entweder explizit in den Modellen enthalten oder entsteht zum Beispiel durch Typ-Instanz-Beziehungen. Eine wissensbasierte Verknüpfung unterschiedlicher Modelle, wie sie unter anderem beim Round-Trip-Engineering mittels Tripel-Graph-Grammatik (TGG) [RGU17] vorkommt, wird in dieser Arbeit nicht behandelt. Sie bietet jedoch die Grundlage, um auch in diesem Bereich eine Nachvollziehbarkeit der Änderungen zu ermöglichen.

Änderungen werden in dieser Arbeit vorrangig auf Ebene der Informations- und Daten-

modelle betrachtet, wodurch eine Abgrenzung zu Ansätzen im Bereich der Evolution von Produktionssystemen, wie unter anderem in [Lad18] und [LFL16] vorgestellt, erfolgt. In diesem Bereich werden Ein- und Ausgangssignale für die Detektion von Veränderungen im Produktionssystem herangezogen, um daraus notwendige Reaktionen oder vorzunehmende Dokumentationen abzuleiten. Die vorliegende Arbeit setzt dementsprechend nachfolgend an, um beispielsweise die durchgeführten Dokumentationen innerhalb der Planungsunterlagen nachvollziehbar zu speichern.

Ein wesentlicher Aspekt bei der Durchführung von Co-Evolutionen und der Speicherung von Änderungen ist die Aggregation der durchgeführten Änderungen zu semantischen Änderungen, die eine Aussage beinhalten, was der Zweck hinter den Änderungen gewesen ist. In diesem Bereich existieren bereits verschiedene Arbeiten, die formale Regelsätze für die Durchführung von Aggregationen bereitstellen. Beispiele hierfür sind unter anderem [Keh15] und [Pap+13]. Auf diese wird innerhalb dieser Arbeit in Bezug auf eine prototypische Implementierung zurückgegriffen.

1.4 Anwendungsfälle

Die Entwicklung der Ansätze und Technologien innerhalb dieser Arbeit erfolgt unter der Betrachtung von zwei Anwendungsfällen. Hierbei handelt es sich jeweils um kollaborative Anwendungen, die zum einen eine hohe Vernetzung und zum anderen hohes Änderungspotential aufweisen. Im Folgenden werden diese jeweils kurz vorgestellt.

1.4.1 Co-Simulation

Im Lebenszyklus einer Prozessanlage spielen Simulationen bereits an vielen Stellen eine wichtige Rolle [OWU14]. Oppelt *et al.* [OWU14] statuieren, dass zukünftig Simulationen direkt in den Anlagenlebenszyklus und einen integrierten Engineeringprozess eingebunden werden können. Notwendige Simulationsmodelle können beispielsweise automatisch aus bestehenden Engineeringdaten generiert werden [Opp+14]. Resultierende Modelle können dann unter anderem für die virtuelle Inbetriebnahme verwendet werden [OU14]. In diesem Zusammenhang fordern Oppelt *et al.* [OWU14] die Spezifikation eines Standards für die Co-Simulation, um die Wiederverwendung von bestehenden Simulationen zu ermöglichen und zu vereinfachen. Neben der Wiederverwendung bietet die Co-Simulation auch Vorteile in Bezug auf die Modellierungseffizienz, da Personen innerhalb eines Projektes zum einen mit den jeweils am besten geeigneten (fachspezifischen) Werkzeugen arbeiten können, zum anderen aber auch auf Basis von bestehenden Präferenzen und Erfahrungen Entscheidungen über das zu verwendende Simulationswerkzeug getroffen werden können. Ein weiterer Vorteil liegt in der Simulation von Wechselwirkungen und Rückkopplungen zwischen den einzelnen Teilkomponenten, die durch eine reine Einzelsimulation nicht erfassbar wären [Smo13]. Durch die Kombination der einzelnen Modelle wird somit eine Gesamtsystemvorhersage ermöglicht. Überdies werden durch die Parallelisierung und Verteilung Beschleunigungsvorteile gegenüber einer integrierten Simulation erwartet, da in einer Co-Simulation die Simulation auf unterschiedliche Geräte verteilt werden kann [Fuj99].

Ein Ansatz für eine Co-Simulationsplattform, die eine Wiederverwendung von bestehenden Simulationen und eine Verschaltung von diesen ermöglicht, wird beispielsweise in [Hen+16a; Hen+16b] vorgestellt. Die Simulationssteuerung erfolgt vereinheitlicht unabhängig vom verwendeten Simulator, um eine nahtlose Interoperabilität zu ermöglichen. Die Co-Simulationsumgebung ist vollständig unter Nutzung von Open Platform Communications Unified Architecture (OPC UA) semantisch beschrieben. Abbildung 1.1 gibt einen Überblick über die allgemeine Architektur des Ansatzes. Diese besteht aus einem zentralen Aggregating Server und generischen Adaptern, die über eine simulatorspezifische Schnittstelle an den zu koppelnden Simulator angeschlossen werden.

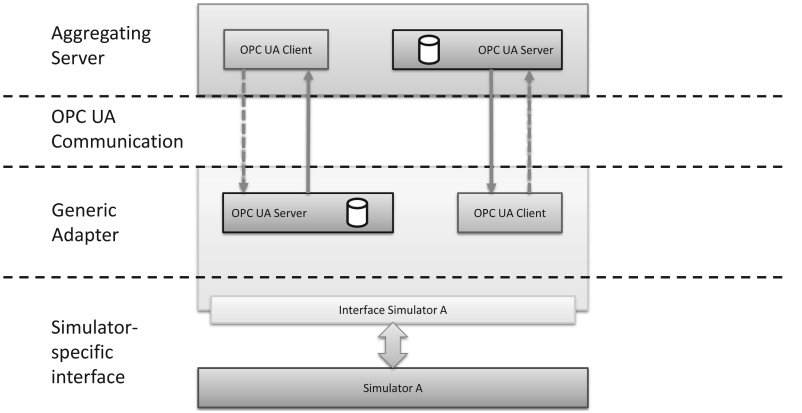


Abbildung 1.1: Architektur der auf OPC UA basierenden Co-Simulationsumgebung [Hen+16a, S. 4]

Zur Durchführung einer Co-Simulation muss die entsprechende Umgebung konfiguriert werden und die zu koppelnden Einzelsimulationen zu einer Gesamtsimulation verschaltet werden. Hierbei muss mit Änderungen an Einzelsimulationen und an den Verschaltungen umgegangen werden, die zusammen mit den Simulationsergebnissen möglichst nachvollziehbar gespeichert werden müssen.

1.4.2 Modularisierung

Neue Herausforderungen in der Prozessindustrie wie immer kürzer werdende Produktlebenszyklen und volatile Märkte erfordern neue Konzepte, um wettbewerbsfähig zu bleiben [LG11]. Die NE 148 [NAM13] formuliert hierfür Anforderungen, die realisiert werden müssen, um diesen Veränderungen begegnen zu können. Insbesondere sind standardisierte Schnittstellen erforderlich, die grundlegend eine modulare Produktion ermöglichen und weiterhin die Wandlungsfähigkeit von modularen Anlagen unterstützen [Bie+16]. Ein wesentlicher Aspekt ist hierbei die Automatisierung einer modularen Prozessanlage. Hierfür werden durch Anwender, Hersteller und Universitäten entsprechend der NE 148 Lösungen

entwickelt. Zentrales Element für die herstellerneutrale Beschreibung von Modulen ist das Module Type Package (MTP). Dieses wurde initial im Dezentrale Intelligenz für modulare Anlagen (DIMA)-Projekt [Hol+14] entwickelt und wird durch Arbeitskreise von Namur, ZVEI und GMA weiterentwickelt und in die Standardisierung überführt [Hen+17]. Abbildung 1.2 gibt einen Überblick, über den dabei verwendeten agilen Prozess der Spezifikation.

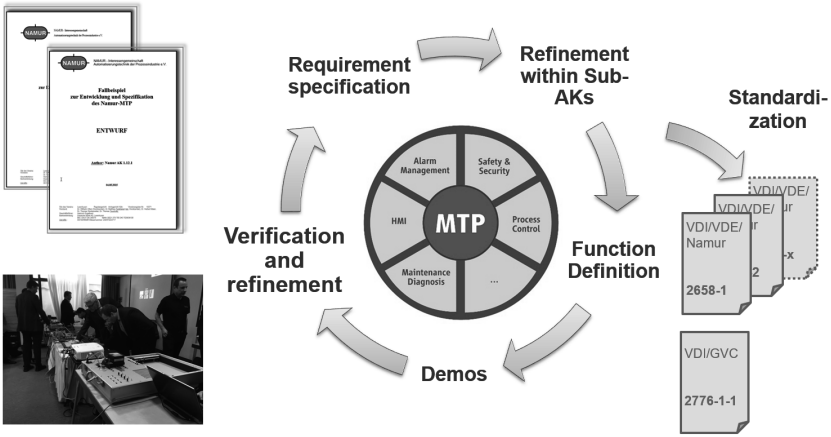


Abbildung 1.2: Agiler Prozess der Spezifikation des MTP [BH17, S. 10]

Das MTP ist dabei in unterschiedliche Aspekte untergliedert, für die in sprintartigen Iterationen Lösungsansätze untersucht und entwickelt werden. Grundlage bildet ein übergeordnetes Lastenheft und konkretisierende Fallbeispiele. Die Ergebnisse der Sprints werden parallel harmonisiert und in die Standardisierung überführt. Die gesammelten Erfahrungen fließen vor der Durchführung der nächsten Iteration wieder in den agilen Prozess zurück. Es handelt sich daher um einen sehr dynamischen Prozess, der davon geprägt ist, dass Änderungen und Erweiterungen an der Spezifikation des MTP vorgenommen und dokumentiert werden müssen. In diesem Zusammenhang spielt auch die Kompatibilität von unterschiedlichen Entwicklungsständen eine sehr große Rolle, da das MTP eine herstellerübergreifende Beschreibung darstellt.

1.5 Gliederung der Arbeit

Die Dissertation gliedert sich im weiteren Verlauf in sieben übergeordnete Abschnitte, die es ermöglichen, die der Arbeit zugrunde gelegten Thesen zu untersuchen. Im Abschnitt 2 erfolgt die Definition von Begrifflichkeiten, die in dieser Arbeit Verwendung finden. Des Weiteren werden die notwendigen Theoriekenntnisse des Themengebiets zusammengefasst.

Im folgenden Abschnitt 3 werden ausgehend von einer Literaturrecherche Anforderungen an das zu konzeptionierende System aufgestellt. Mit Hilfe der Anwendungsfälle werden die Anforderungen bestätigt beziehungsweise erweitert, um ein möglichst vollständiges Bild zu erhalten. Durch die Spiegelung der Anforderungen an bestehende Ansätze können die Lücken aufgedeckt werden, die im weiteren Verlauf der Arbeit geschlossen werden sollen. Die Zusammenfassung der Analyseergebnisse und eine entsprechende Priorisierung der Anforderungen ermöglicht im folgenden Abschnitt 4 die Fokussierung auf die wesentlichen zu entwickelnden Komponenten. Ausgehend vom Lebenszyklus werden die Anforderungen in ein Konzept für ein RMS überführt. Die jeweiligen Komponenten dieses Systems werden mathematisch und semantisch beschrieben, um ein technologieunabhängiges Konzept zu erhalten, das sich auf unterschiedliche Technologien übertragen lässt. In der Implementierung, vorgestellt in Abschnitt 5, wird das technologieunabhängige Konzept beispielhaft für eine spezifische Technologie, das Semantic Web, umgesetzt. Durch diese Umsetzung wird die Funktionsweise des theoretisch beschriebenen Konzeptes nachgewiesen und anschließend im Abschnitt 6 verifiziert. Die Verifikation erfolgt auf der einen Seite anhand von Testfällen der Implementierung, aber auch durch den theoretischen Nachweis der formalen Beschreibungen anhand von Beispielen und dem Anwendungsfall der Co-Simulation. Schließlich werden die Ergebnisse der Arbeit in Abschnitt 7 kritisch diskutiert und den aufgestellten Thesen gegenübergestellt. Die Zusammenfassung der Dissertation erfolgt im Abschnitt 8, wobei ebenfalls ein Ausblick auf Anknüpfungspunkte für Folgearbeiten gegeben wird.

2 Grundlagen

In diesem Abschnitt werden zunächst die grundlegenden Begriffsdefinitionen und Theoriekenntnisse des Themengebietes zusammengefasst. Hierbei werden zu Beginn die Aspekte der Veränderlichkeit auf einer allgemeinen Ebene dargestellt, bevor im nächsten Abschnitt in die Informationsmodellierung eingeführt wird. Über ihren Lebenszyklus hinweg unterliegen diese Modelle einer natürlichen Evolution, deren Begrifflichkeiten im Folgenden vorgestellt werden. Da während der Evolution eine Nachvollziehbarkeit der durchgeführten Änderungen gewährleistet werden muss, wird nachfolgend in die Grundlagen der Revisionsverwaltung eingeführt. Abgeschlossen wird dieser Abschnitt mit einer Beschreibung von Konsistenz, da diese sowohl während der Evolution, aber auch innerhalb der Revisionsverwaltung von essenzieller Bedeutung ist.

2.1 Aspekte der Veränderlichkeit

Für zukünftige Systeme existieren nach Fricke und Schulz [FS05] drei Treiber. Hierzu zählen die Dynamik der Märkte, die technologische Weiterentwicklung und die Vielfalt der Umgebungen. Durch die immer größere Dynamik in den Märkten verringert sich die Zeitspanne zwischen dem Design Freeze und der Auslieferung der Systeme. Die Architekturen müssen dabei auch nach einer Einführung veränderbar bleiben, da sich die Umgebung weiterentwickelt und daraus wiederum neue Anforderungen an die Architekturen entstehen. Fricke und Schulz [FS05] stellen dabei fest, dass der Erfolg einer Architektur von der Umsetzbarkeit der neuen Anforderungen abhängt. Ein weiterer Punkt der Dynamik ist die immer weiter fortschreitende Individualisierung der Produkte. Unternehmen müssen hierbei Standardkomponenten entwickeln und wiederverwenden, um die Wirtschaftlichkeit und den Erfolg des Unternehmens sicherzustellen. Der zweite Treiber ist dadurch gekennzeichnet, dass sich Systeme und deren Funktionen schnell weiterentwickeln müssen, da sich die Lebensdauer der Produkte zunehmend verkürzt und die korrespondierenden Systeme darauf reagieren müssen. Daraus resultiert, dass die Wettbewerbsfähigkeit der Unternehmen stark von der Verwendung von neuen Technologien abhängt. Der letzte von Fricke und Schulz [FS05] aufgeführte Treiber zielt auf die Vielfalt der Umgebungen ab, da komplexe Systeme eine Komposition aus unterschiedlichen Technologien sind und diese Systeme oft wiederum in höhere Systeme integriert sind. Die jeweils beteiligten Komponenten stehen dabei in Wechselbeziehungen und beeinflussen sich gegenseitig.

Daraus resultieren die in Abbildung 2.1 dargestellten Aspekte der Veränderlichkeit. Systemarchitekturen müssen nach [FS05] einerseits die Fähigkeit der einfachen und schnellen Weiterentwicklung enthalten und andererseits auch die Fähigkeit besitzen, unempfindlich beziehungsweise anpassungsfähig auf sich ändernde Umgebungsbedingungen zu reagieren.

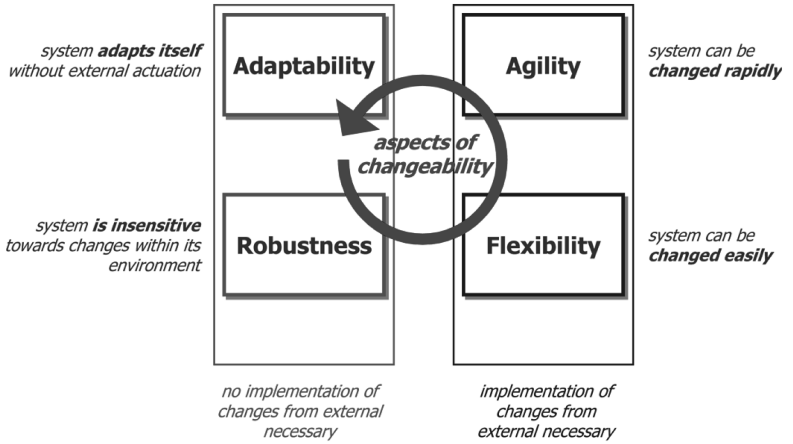


Abbildung 2.1: Aspekte der Veränderlichkeit [FS05, S. 347]

Bei der Produktentwicklung werden Änderungen zumeist entweder verhindert oder vorgezogen, da Änderungen in späteren Phasen der Entwicklung zu hohen Kosten führen [FS05; Fri+00]. Jedoch lassen sich auch in späteren Phasen Änderungen nicht vermeiden [FS05], weshalb eine Architektur benötigt wird, die mit Änderungen über den gesamten Lebenszyklus hinweg umgehen kann. Bei einer solchen Architektur spielen aber wiederum auch die Kosten für die Weiterentwickelbarkeit im Vergleich zu den Kosten durch zusätzliche Änderungen eine wesentliche Rolle. Abbildung 2.2 zeigt den Grad der Veränderlichkeit im Vergleich zu den Änderungskosten und die aufsummierten Gesamtkosten, die am Schnittpunkt der beiden aufsummierten Kostenkurven ein Minimum besitzt, auf das bei der Entwicklung von Systemarchitekturen abgezielt werden sollte.

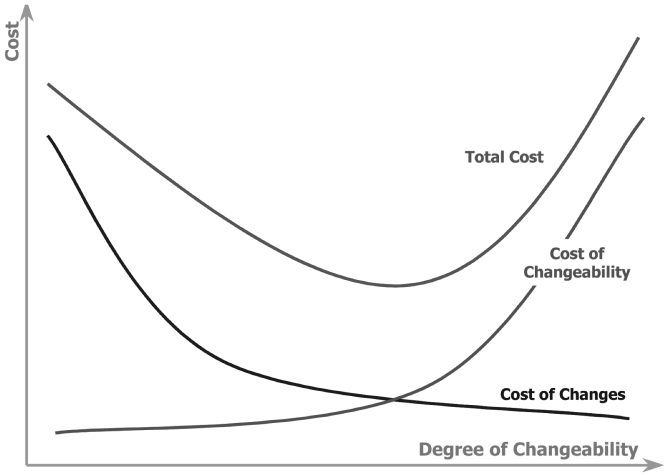


Abbildung 2.2: Grad der Veränderlichkeit vs. Kosten [FS05, S. 356]

2.2 Informationsmodellierung

Informationsmodelle sind für eine durchgehende Digitalisierung und die Realisierung von komplexen Systemen unabdingbar. Sie unterliegen wie die Systeme selbst einem Lebenszyklus und müssen daher mit einhergehenden Änderungen umgehen können. Informationsmodelle sind dabei für die Nutzer oft nicht einsehbar, da sie von den Systemen nur im Hintergrund benutzt werden. In diesem Abschnitt werden die in der Arbeit verwendeten Begrifflichkeiten in Bezug auf die Informationsmodellierung definiert und es wird auf den Lebenszyklus von Modellen sowie deren Vernetzung eingegangen.

2.2.1 Terminologie

Ausgehend von Informationsmodell und Semantik erfolgt im Weiteren eine Definition von wichtigen Begrifflichkeiten wie Informationsraum und Ontologie. Schließlich werden Arten von Informationsmodellen sowie deren Einordnung in eine Metadatenarchitektur vorgestellt.

2.2.1.1 Informationsmodell und Semantik

Beim Austausch von Daten zwischen unterschiedlichen Informationsträgern muss zum einen ein gemeinsamer Übertragungsweg definiert und zum anderen festgelegt werden, wie die Interpretation der Daten zu erfolgen hat [Gra16]. Hierfür sind die beiden Begriffe Syntax und Semantik von wesentlicher Bedeutung. Durch die Syntax von Ausdrücken wird deren formale Korrektheit definiert [Gra16]. Sie gibt demnach eine Menge von Regeln

vor, nach denen diese Ausdrücke gebildet werden dürfen [Tan06]. Durch die Semantik hingegen wird den Ausdrücken eine entsprechende Bedeutung zugewiesen [Gra16].

Aus der Sprachwissenschaft heraus wird Semantik allgemein „als die Erforschung der Bedeutung“[Lyo80, S. 15] definiert. Nach Lyons [Lyo80] wird die Bedeutung von Wörtern sowie Sätzen durch den Gebrauch von Sprache in kommunikativen Situationen gelernt beziehungsweise beibehalten. In diesem Zusammenhang spielen auch die Benennung und die Referenz eine wesentliche Rolle. Die Benennung erfolgt dabei durch den Rückgriff auf Zeichen oder Symbole, die zusammengesetzt ein Wort ergeben, dem dann wiederum eine Bedeutung zugeordnet werden kann. Durch Referenzen kann dieses Wort dann wiederum in einen Kontext eingebettet werden. Lyons [Lyo80] geht weiterhin auf die Bereiche Klassenlogik, Referenz, Objekte mit deren Eigenschaften und Relationen sowie Sinnrelationen, wie Opposition und Kontrast, Hierarchien oder Teil-Ganzes-Beziehungen ein, die in einem engen Zusammenhang zur Bedeutung stehen und den Kontext bereitstellen, in dem die Betrachtung der Bedeutung zu erfolgen hat. Diese Eigenschaften finden sich auch in der Informationstechnik wieder, um die Semantik von Daten zu beschreiben, wie beispielsweise in [She97] dargestellt. Im Rahmen dieser Arbeit werden für die formale Beschreibung der Bedeutung von zugrundeliegenden Daten und für die Beschreibung des zugehörigen Kontextes Informationsmodelle verwendet, wie im Folgenden definiert. Der Kontext ergibt sich dabei unter anderem aus zugehörigen Eigenschaften, Relationen oder Attributen. Die Repräsentation der Informationsmodelle und der sich daraus ergebenden Beschreibung der Semantik erfolgt durch UML-Modelle und eine textuelle Beschreibung, wie die Benennung zu interpretieren ist.

„Ein Modell ist eine Repräsentation eines Systems von Objekten, Beziehungen und/oder Abläufen. Ein Modell vereinfacht und abstrahiert dabei im allgemeinen [sic!] das repräsentierte System.“[Kön12, S. 25] Kastens und Büning [KB14] definieren weiterhin, dass jeweils nur spezielle Aspekte eines Originals durch ein Modell beschrieben werden. Die wesentlichen Elemente hierbei sind die Struktur, die die Zusammensetzung aus Bestandteilen kennzeichnet, die Eigenschaften, die Teile des Originals beschreiben, die Beziehungen, die zwischen Teilen des Originals vorhanden sind und das Verhalten, das das Original bei der Anwendung von Operationen zeigt. Die Formulierung der Modelle soll nach [KB14] möglichst deklarativ oder deskriptiv erfolgen.

Majer [Maj10] formuliert nach [Her05], dass ein Informationsmodell einen Wirklichkeitsausschnitt formalisiert. In diesem entstehenden Diskursraum werden Einschränkungen in Bezug auf die Semantik zu Zwecken der Kommunikation vorgenommen [Gra16]. Nachfolgend sind die Definitionen für den Begriff Informationsmodell nach Westerinen und Lee aufgeführt, die zueinander ergänzend sind. So werden in [Lee99] die Einzelbestandteile ausführlich aufgeschlüsselt und in [Wes+01] erfolgt die Ergänzung der unabhängigen Beschreibung.

> Ein Informationsmodell ist eine Repräsentation von Konzepten, Beziehungen, Einschränkungen, Regeln und Operationen, um für eine gewählte Domäne des Diskurses die Datensemantik zu spezifizieren. Der Vorteil für die Verwendung eines Informationsmodells liegt darin, dass es eine gemeinsam nutzbare, stabile und organisierte Struktur von Informationsanforderungen für den

> Domänenkontext bereitstellen kann.[1)] (nach [Lee99, S. 1])
>
> Ein Informationsmodell ist eine Abstraktion und Repräsentation von Entitäten in einer verwalteten Umgebung, ihrer Eigenschaften, Attribute und Operationen sowie deren Beziehungen untereinander. Es ist unabhängig von einem bestimmten Repository, Software-Nutzung, Protokoll oder Plattform.[2)] (nach [Wes+01])

Auf Basis der vorangehenden Definitionen wird für diese Arbeit die nachfolgende Definition abgeleitet, die die wesentlichen Aspekte der vorangehenden Definitionen aufgreift und zusammenfasst. Grundlage bildet die Definition von Lee mit dem Zusatz der Definition von Westerinen sowie die Formalisierung durch Majer nach Hermsdorfer.

> ***Definition Informationsmodell:***
> *Ein Informationsmodell formalisiert einen Wirklichkeitsausschnitt. Es handelt sich dabei um eine Repräsentation von Konzepten, Beziehungen, Einschränkungen, Regeln und Operationen, um für eine gewählte Domäne des Diskurses die Datensemantik zu spezifizieren. Das Informationsmodell stellt dabei eine gemeinsam nutzbare, stabile und organisierte Struktur von Informationsanforderungen für den Domänenkontext bereit und ermöglicht eine von konkreten Umsetzungen unabhängige Beschreibung.*

2.2.1.2 Informationsraum

Ein weiterer wichtiger Begriff mit Bezug zu Informationsmodellen ist der Informationsraum. In dieser Arbeit wird der von Graube [Gra16] gewählten Definition nach Hilbert [Hil15] gefolgt. Diese fasst die semantische Beschreibung sowie die daraus resultierende Möglichkeit der formalen Interpretation zusammen und beschreibt des Weiteren die semantischen Verknüpfungen [Gra16], auf die im Abschnitt 2.2.3 detaillierter eingegangen wird.

> ***Definition Informationsraum:***
> *Die Informationsmodelle bilden zusammen mit den Daten einen Informationsraum als „[...] semantisch beschriebene [...] Menge instanziierter oder referenzierter Informationsressourcen sowie zugehöriger semantischer Verknüpfungsinformationen"* [Hil15, S. 63].

[1)] Übersetzung des Autors aus dem Englischen: „An information model is a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. The advantage of using an information model is that it can provide sharable, stable, and organized structure of information requirements for the domain context."[Lee99, S. 1]

[2)] Übersetzung des Autors aus dem Englischen: „An abstraction and representation of the entities in a managed environment, their properties, attributes and operations, and the way that they relate to each other. It is independent of any specific repository, software usage, protocol, or platform."[Wes+01]

2.2.1.3 Arten von Informationsmodellen

Informationsmodelle können nach [VDI16] in zwei Arten unterschieden werden. Das abstrakte Informationsmodell nimmt dabei eine Abstraktion eines relevanten Ausschnitts der Realität vor, wobei dieses nicht notwendigerweise formalisiert sein muss. Es dient damit vorrangig der Konzepterstellung und Dokumentation, da es von Systemen nicht automatisiert auswertbar ist. Für eine solche automatisierte Auswertung muss erst eine Abbildung auf eine konkrete Technologie erfolgen. Das daraus resultierende konkrete Informationsmodell ist dann formalisiert in einer Zieltechnologie beschrieben und umgesetzt.

Neben den Informationsmodellen existieren des Weiteren Datenmodelle, die die durch die Informationsmodelle bereitgestellte Semantik nutzen. Datenmodelle tragen daher selbst keine Beschreibung der Semantik und werden nur für die Übertragung von Daten genutzt. Durch die Referenzierung eines entsprechenden Informationsmodells durch das Datenmodell können jedoch die Daten beim Empfänger semantisch korrekt interpretiert werden. Die Entitäten der Datenmodelle stellen damit Instanzen von Konzepten dar, die in korrespondierenden Informationsmodellen definiert werden. Im Gegensatz dazu sind Informationsmodelle nicht auf die Definition von Konzepten beschränkt und können darüber hinaus ebenso Instanzen enthalten.

Die Konzepte der Meta Object Facility (MOF) haben neben der allgemeinen Unterscheidung in abstrakte und konkrete Informationsmodelle eine weite Verbreitung gefunden. Die zugehörige Metadatenarchitektur findet vorrangig im Model-driven Engineering (MDE) Anwendung und besteht aus den nachfolgend aufgeführten vier Ebenen [OMG02]:

- M0 – Konkrete Daten
  Beschreiben die Instanz des Nutzermodells.
- M1 – Nutzermodell
  Dieses Modell entspricht den Vorgaben des Metamodells.
- M2 – Metamodell
  Hierdurch wird die „Sprache“ der Nutzermodelle spezifiziert. Ein Beispiel ist die Unified Modeling Language (UML).
- M3 – Metametamodell
  Spezifizieren wiederum die „Sprache“ der Metamodelle.

Die allgemeine Unterscheidung kann dabei ebenso in die Ebenen der MOF eingeordnet werden. Das Datenmodell wird zur Bereitstellung der Instanzen auf der Ebene M0 verwendet. Sowohl für das abstrakte als auch für das konkrete Informationsmodell gilt, dass diese der Ebene M1 zugeordnet werden können. Für das konkrete Informationsmodell existieren über der Ebene M1 des Weiteren die Ebenen M2 und M3 mit Metamodell und Metametamodell. Diese beiden Ebenen existieren für das abstrakte Informationsmodell nur, wenn dieses in einer formalen Art und Weise vorliegt und mittels Metamodellen beschrieben werden kann.

2.2.1.4 Ontologie

Ein Informationsmodell bietet nach Graube [Gra16] die Grundlage, um „[...] durch eine formale und explizite Definition eine Ontologie erstellen [...]“[Gra16, S. 15] zu können. Vereinfacht kann eine Ontologie als „[...] formale Definition von Begriffen und deren Beziehungen als Grundlage für ein gemeinsames Verständnis“[Bus+14, S. 286–287] beschrieben werden. Ontologien ermöglichen daher, dass die Semantik von Information in einer maschinenlesbaren Art und Weise zwischen unterschiedlichen Akteuren (Maschinen und Menschen) kommuniziert werden kann [Fen01].

Ontologien finden unter anderem im Semantic Web Anwendung. Die Semantic Web Technologien sind dabei ein Teil der *W3C Data Activity* [W3C]. Nach [W3C] ist das Ziel dieser Aktivität, dass Personen und Organisationen in die Lage versetzt werden, Daten mithilfe ihrer vorhandenen Werkzeuge und aus bestehenden Arbeitsmethoden heraus teilen zu können. Dabei liegt der Fokus darauf, dass dies in einer Art und Weise geschieht, die es anderen ermöglicht, Werte abzuleiten und hinzuzufügen und diese entsprechend zu nutzen. Es steht hierbei nicht nur die Interoperabilität von Daten, sondern auch von Communities im Mittelpunkt. Basistechnologie des Semantic Web ist das Resource Description Framework (RDF), das gleichzeitig das Datenmodell für das Semantic Web darstellt. Für die Erstellung von Ontologien stehen wiederum als Basis das Resource Description Framework Schema (RDFS)[3)] und die Web Ontology Language (OWL)[4)] zur Verfügung [Gra16].

2.2.2 Lebenszyklus

Über den Lebenszyklus eines Systems hinweg unterliegen die verwendeten Modelle unterschiedlichen Änderungen. Dabei lässt sich die Faustregel anwenden, dass sich untere Ebenen der MOF bedeutend häufiger ändern als überlagerte. Konkrete Daten (M0) ändern sich daher oft bei jeder Interaktion mit den Modellen. Eine Erweiterung oder Anpassung des Informationsmodells auf Ebene M1 wird jedoch nur auftreten, wenn sich die Anforderungen an das System ändern, wodurch Funktionsanpassungen des Gesamtsystems vorgenommen werden müssen. Ein Wechsel der Sprache (M2) wird dahingegen ab einem bestimmten Reifegrad nur noch sehr selten vorkommen. Änderungen auf dieser Ebene sind mit einem enorm hohen Aufwand verbunden, wobei die Notwendigkeit eines solchen Wechsels aufgrund von hohen Abstraktionsmechanismen zumeist auch nicht gegeben ist.

Bereits an dieser Stelle lässt sich feststellen, dass Modelle, ebenso wie Systeme und Produkte im Allgemeinen, einen Lebenszyklus aufweisen. Die aus beispielsweise Wartung und Pflege heraus entstehenden Änderungen müssen dabei wiederum nachvollziehbar gespeichert werden, damit diese für folgende Wartungen als Grundlage bereitstehen.

[3)] https://www.w3.org/TR/rdf-schema/ (besucht am 29.11.2020)
[4)] https://www.w3.org/OWL/ (besucht am 29.11.2020)

2.2.3 Vernetzung innerhalb eines Informationsraums

Informationsräume sind durch eine Vernetzung gekennzeichnet. Es bestehen auf der einen Seite vertikale Beziehungen (in Bezug auf die Ebenen des MOF), wie unter anderem Typ-Instanz-Beziehungen oder Abhängigkeiten zwischen Nutzer- und Metamodell. Auf der anderen Seite bestehen aber auch Relationen innerhalb der konkreten Daten, die durch ein Nutzermodell beschrieben werden, oder auch Abhängigkeiten, die zwischen den unterschiedlichen Modellen auf höheren Ebenen des MOF auftreten können. Der damit einhergehende Begriff der verbindungsorientierten Modelle wird vor allem im Bereich von Netzwerken verwendet [Wal10; Van91]. Dabei werden Verbindungen zwischen unterschiedlichen Teilnehmern hergestellt, um im Folgenden Daten übertragen zu können. Im Endeffekt handelt es sich dabei jeweils um die Beschreibung einer Topologie von Komponenten und deren Verbindungen untereinander, wie beispielsweise in [ONF16; Van91; Int+19] dargestellt. Neben dem Bereich der Netzwerke existieren weitere Anwendungsgebiete, in denen diese Begrifflichkeit zumeist nicht explizit aufgeführt wird. Beispiele hierfür sind untere anderem die Beschreibung von Bedienbildern in modularen Anlagen [VDI18], Co-Simulationen [Hen+16b], Flowsheets, Matlab Simulink Projekte und Kontaktpläne [DIN14].

Im weiteren Verlauf werden diese unterschiedlichen, bei der Modellierung und der späteren Verwendung auftretenden Relationen allgemein als Vernetzung in den Informationsräumen bezeichnet. Der Begriff der Vernetzung geht dabei auch mit der Natur der Modelle einher, da diese als Graph interpretiert werden können [Lev+10]. Sie lassen sich demnach als eine Menge an Knoten und Kanten beschreiben.

2.3 Evolution

Informationsmodelle unterliegen einer natürlichen Evolution während ihres Lebenszyklus. So werden beispielsweise Planungsdaten während des Engineerings stetig weiterentwickelt oder müssen aufgrund von Anforderungsänderungen beziehungsweise durchzuführenden Fehlerbehebungen angepasst werden [Vog+15b]. Im Folgenden werden die in dieser Arbeit verwendeten Begrifflichkeiten definiert und die Verwendung von Evolution in verwandten Themengebieten vorgestellt.

2.3.1 Terminologie

Die Begriffe Evolution, Evolvability und Wartbarkeit stehen in einem engen Zusammenhang [RB09], weshalb eine Unterscheidung und Definition der einzelnen Begrifflichkeiten notwendig ist. In der Literatur werden die Begriffe jedoch oft ohne explizite Definition verwendet, was aufgrund des Interpretationsspielraums zu Missverständnissen führen kann. Für die weitere Arbeit werden daher an dieser Stelle Definitionen aus der bestehenden Literatur aufgegriffen und gegebenenfalls erweitert, sodass sie dieser Arbeit zugrunde gelegt werden können.

2.3.1.1 Evolution

Evolution wird vor allem im Zusammenhang mit Softwaresystemen [RLL98], jedoch auch in Bezug auf Produktionssysteme [Lad18], definiert. Die Definitionen, die beispielsweise in [RLL98], [RB09] oder [Ruh+14] gegeben werden, haben stets gemein, dass schrittweise und kontinuierliche Änderungen beschrieben werden. Davon ausgehend wird im Folgenden als Basis die Definition von Ruhroth [Ruh+14] verwendet, da diese zusätzlich den Gedanken der Transformation und der Begründung von Änderungen aufgreift.

> Der Begriff der Evolution kann als anhaltende und schrittweise Änderung von Entwicklungsartefakten definiert werden. Dabei behält jeder Schritt die meisten Eigenschaften (Funktionalität und Sicherheit) des vorherigen Systems bei und wird durch eine Begründung gerechtfertigt. Diese Evolutionsschritte können als Transformation von Modellen aus ihrem aktuellen Status in einen modifizierten betrachtet werden.[5] (nach [Ruh+14, S. 1])

Des Weiteren beziehen Riebisch und Bode [RB09] nach Lehmann [Leh80] den Begriff der Evolution auf den gesamten Lebenszyklus eines Softwaresystems. Dies bedeutet, dass von der initialen Entwicklung über die Wartung bis zum Reengineering alle Phasen eingeschlossen sind [RB09]. Diese Erweiterung fließt in die Definition von Ruhroth ein, woraus die nachstehende Definition folgt, die im Weiteren der Arbeit zugrunde gelegt wird.

> ***Definition Evolution:***
> *Der Begriff der Evolution kann als anhaltende und schrittweise Änderung von Entwicklungsartefakten definiert werden, die während des gesamten Lebenszyklus von Modellen stattfinden. Dabei behält jeder Schritt die meisten Eigenschaften (Funktionalität und Sicherheit) des vorherigen Systems bei und wird durch eine Begründung gerechtfertigt. Diese Evolutionsschritte können als Transformation von Modellen aus ihrem aktuellen Status in einen modifizierten betrachtet werden.*

2.3.1.2 Co-Evolution

Von Evolution im Allgemeinen abgeleitet existiert in der Literatur weiterhin der Begriff der Co-Evolution (oder auch als Coupled-Evolution bezeichnet [HW14]), der unter anderem im Bereich des MDE Verwendung findet. Hierunter wird die Anpassung eines Modells auf Basis von Änderungen an einem korrespondierenden Modell verstanden. Beispiele für Co-Evolutionen und die zugrunde liegenden Modellrelationen können zwischen Metamodellen und Nutzermodellen [DIP11], zwischen Typ (Nutzermodell) und Instanz (konkrete

[5] Übersetzung des Autors aus dem Englischen: „The term evolution can be defined as the ongoing change of development artifacts in a stepwise manner, such that every step preserves most properties (functionality and security) of the former system and is justified by a rationale. This evolution steps can be seen as a transformation of models from their current state into a modified one.“[Ruh+14, S. 1]

Daten) [DIP11] oder Modell und Modell auf unterschiedlichen Abstraktionsniveaus (zum Beispiel Nutzermodell und Nutzermodell) [HHH14; Ruh+14] bestehen. Je nach Art der zugrunde liegenden Abhängigkeit und der Definition des konsistenten Gesamtzustands (Begriff der Konsistenz wird in Abschnitt 2.5 erläutert) wird die Co-Evolution unidirektional oder bidirektional durchgeführt. So werden Änderungen an Typmodellen zumeist nur auf die Instanzdaten angewendet und nicht umgekehrt. Änderungen an Modellen unterschiedlicher Abstraktion sollten jedoch zumeist wechselseitig angeglichen werden, um einen konsistenten Gesamtzustand zu erhalten. Für Co-Evolution lässt sich nachstehende Definition ableiten, die im weiteren Verlauf der Arbeit Anwendung findet.

> ***Definition Co-Evolution:***
> *Co-Evolution bezeichnet die Anpassung eines Modells auf Basis der Evolution eines korrespondierenden Modells unter Nutzung von bestehenden Modellrelationen. Je nach Art der zugrunde liegenden Abhängigkeit und der Definition des konsistenten Gesamtzustands wird die Co-Evolution unidirektional oder bidirektional durchgeführt.*

Abbildung 2.3 stellt anhand eines Beispiels aus [Ruh+14] die Beziehung zwischen Evolution und Co-Evolution dar. Durchgeführte Änderungen an dem aufgeführten Security Maintenance Model haben Auswirkungen auf das zugehörige System Model. In einem abstrakteren Beispiel könnte es sich dabei um ein Nutzermodell und konkrete Daten handeln. Auf Basis der Evolution des Nutzermodells werden notwendige Evolutionen auf den konkreten Daten abgeleitet, die dann mittels der Co-Evolution auf die konkreten Daten angewendet werden.

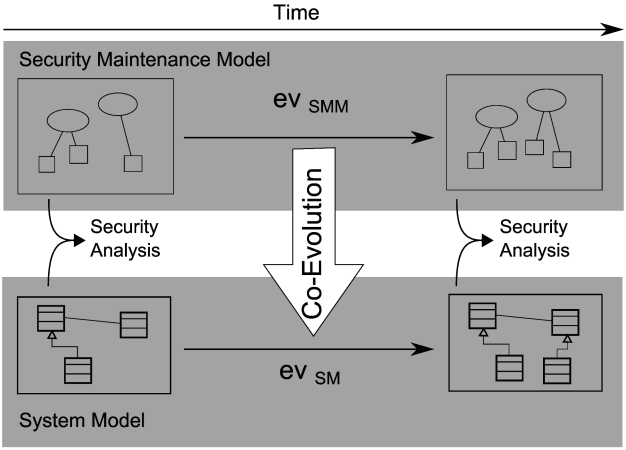


Abbildung 2.3: Beispiel für die Beziehung zwischen Evolution und Co-Evolution [Ruh+14, S. 1]

2.3.1.3 Evolvability

Evolvability beschreibt allgemein den Zustand eines Systems, der dadurch gekennzeichnet ist, dass er schnell und einfach Änderungen erlaubt. Evolvability lässt sich daher in etwa mit Weiterentwickelbarkeit übersetzen [RB09]. Der Unterschied zur Changeability (Veränderlichkeit), wie in Abschnitt 2.1 dargestellt, ist in der Literatur zumeist nicht trennscharf und wird oft synonym verwendet. Es existieren jedoch auch Unterscheidungen, wie in Beesemyer [Bee12] dargestellt. Beesemyer charakterisiert dabei Evolvability als ein Subset der Changeability, wobei Evolvability die Fähigkeit einer Architektur beschreibt, vererbt und über Generationen hinweg weiterentwickelt zu werden. Brinca [BBR09] hingegen stellt Changeability als eine Verfeinerung der Evolvability dar. Ein von Rowe *et al.* [RLL98] durchgeführter Vergleich von Definitionen kommt zu dem Schluss, dass eine Definition von Evolvability nicht ausreichend ist und es sich bei Evolvability um eine zusammengesetzte Qualität handelt.

> Evolvability: Ein Attribut, das sich auf die Fähigkeit eines Systems bezieht, über den gesamten Lebenszyklus hinweg Anforderungsänderungen aufnehmen zu können, während die Kosten so gering wie möglich gehalten werden und die architektonische Integrität erhalten bleibt.[6] (nach [RLL98, S. 5])

Die Definition von Rowe kennzeichnet zum einen, dass die Architektur mit Änderungen kosteneffektiv umgehen kann und zum anderen, dass die Integrität der Architektur dabei gewährleistet ist. Dies wird auch durch [BCE07] bestätigt, wobei die Definition von Breivold *et al.* expliziter auf die Stimuli von Änderungen in Softwaresystemen eingeht. Ebenso bestätigen Bahill und Botta [BB08] nach Christian und Olds [CO05] die Definition von Rowe. Die Basis für die in dieser Arbeit verwendeten Definition bildet demnach Rowe mit der Ergänzung des Generationsgedankens von Beesemyer, wie nachfolgend aufgeführt.

> ***Definition Evolvability:***
> *Evolvability kennzeichnet die Fähigkeit einer Architektur, mit möglichst geringen Kosten und unter Beibehaltung der architektonischen Integrität, auf Anforderungsänderungen über den gesamten Lebenszyklus hinweg reagieren zu können und über mehrere Generationen weiterentwickelt zu werden.*

Riebisch [RB09] führt in diesem Zusammenhang Prinzipien auf, die die Evolvability beeinflussen. Hierzu gehören einerseits positive Einflüsse wie Abstraktion und Modularität, die es ermöglichen die Komplexität zu beherrschen, aber andererseits auch negative Einflüsse wie Komplexität und Kopplungen. Negative Einflüsse können jedoch zumeist nicht beeinflusst werden, wenn sich die Anforderungen nicht anderweitig umsetzen lassen. Brcina *et al.* [BBR09] geben hierzu einen Überblick über die beinhalteten Prinzipien, deren Zusammenhänge und Einflüsse aufeinander, sowie der Zuordnung zu Qualitätsattributen, wie in Abbildung 2.4 dargestellt.

[6] Übersetzung des Autors aus dem Englischen: „Evolvability: An attribute that bears on the ability of a system to accommodate change in its requirements throughout the system's lifespan with the least possible cost while maintaining architectural integrity.“[RLL98, S. 5]

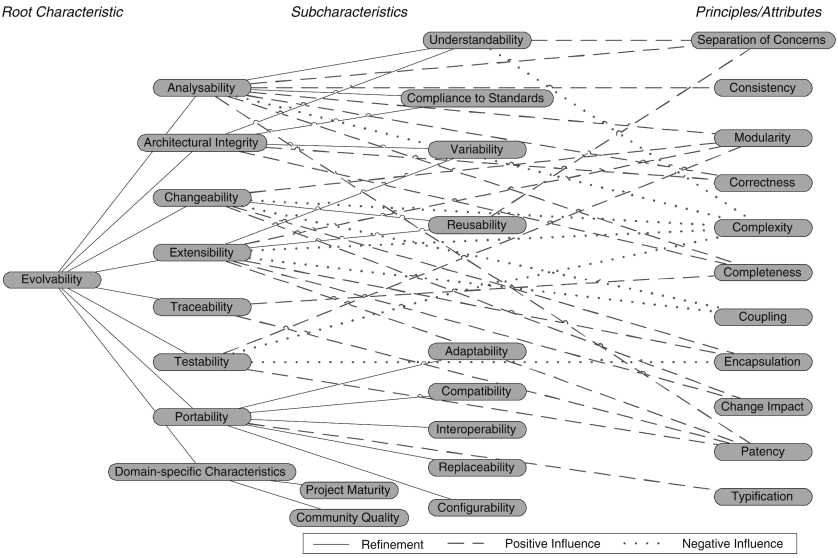


Abbildung 2.4: Charakteristiken und Attribute mit Einfluss auf Evolvability (Ausschnitt aus [BBR09, S. 198])

2.3.1.4 Wartung und Wartbarkeit

In der Softwareentwicklung kennzeichnet Wartung die Modifikation von Softwareprodukten, um Anpassungen an veränderte Anforderungen vorzunehmen, nachdem die Software ausgeliefert worden ist [RB09] nach [IEE98]. Bei der Wartung werden typischerweise keine bedeutenden Änderungen an der Architektur vorgenommen. Evolution hingegen schließt im Gegensatz dazu die Wartung sowie größere Änderungen über den gesamten Lebenszyklus hinweg mit ein [Men17]. Wartbarkeit entspricht einem Qualitätsmerkmal und wird durch die ISO-Norm 9126 [ISO01] definiert. Dabei bezieht sich Wartbarkeit auf den Aufwand, der sich aus Änderungen ergibt [RB09].

2.3.2 Evolution in verwandten Themengebieten

Die Problemstellung der Evolution wird neben der Biologie auch in unterschiedlichen Softwaredomänen adressiert. Nachfolgend werden die Schema-Evolution und die Ontologie-Evolution als bekannteste Vertreter vorgestellt und miteinander verglichen.

2.3.2.1 Schema-Evolution

Objektorientierte Datenbanken Die bekannteste Domäne für Evolution in Softwaresystemen ist die objektorientierte Datenbank mit der sogenannten Schema-Evolution [Lev+10]. Durch Änderungen an einem Schema kann die Funktionsweise von abhängigen Objekten beeinträchtigt werden. Bei diesen Objekten handelt es sich beispielsweise um Views der Datenbank, um Abfragen auf die Datenbank, zu erfüllende Bedingungen in der Datenbank, auf das Schema referenzierende Programme oder im Allgemeinen die Datenbank als Instanz des Schemas [BM07]. Änderungen am Schema müssen dementsprechend korrekt und effizient an die abhängigen Objekte propagiert werden [RB06]. Ansätze für den Umgang mit Schemaänderungen erstellen zumeist ein Mapping zwischen den evolvierten Schemas und leiten daraus die durchzuführenden Modellmanagementoperationen zur Anpassung des Mappings zwischen abhängigen Objekten und evolviertem Schema ab [BM07]. Resultierende Ansätze werden beispielsweise in [BM07; Ban+87; RR97] beschrieben. Eine umfassende Sammlung und Kategorisierung von bestehenden Veröffentlichungen in diesem Themenbereich wird in [RB06] vorgestellt.

Softwareengineering Im Softwareengineering lassen sich viele der Problemstellungen der Schema-Evolution wiederfinden. Diese kommen vor allem dann zum Tragen, wenn neue Softwareversionen erstellt werden. Durch die Abhängigkeiten und Beziehungen zwischen einzelnen Programmteilen müssen bei einer Evolution von beispielsweise Programmschnittstellen oder Klassenhierarchien die Auswirkungen auf die abhängigen Teile propagiert werden [RB06]. Viele der Veröffentlichungen in diesem Bereich beziehen sich dabei auf objektorientierte Softwareentwicklung. In diesem Zusammenhang werden auch Ansätze für Modell Evolution vorgestellt, wie beispielsweise in [Lev+10] präsentiert. Weiterhin formulierten Lehmann und Belady für die Software-Evolution im Allgemeinen eine Reihe von Gesetzmäßigkeiten, die über die Jahre selbst evolvierten und von Lehmann weiterentwickelt wurden [Leh96; Leh80; LR03]. Des Weiteren werden Herausforderungen für die Software-Evolution im Zusammenhang mit automatisierten Produktionssystemen von Vogel-Heuser *et al.* in [VR15; Vog+15a; Vog+15b] dargestellt. Überdies werden auch Ansätze in der Literatur beschrieben, die sich mit der Bewertung und dem Vergleich von Softwaresystemen in Bezug auf Evolvability beschäftigen. Ein Beispiel hierfür ist in [BCE08] dargestellt.

2.3.2.2 Ontologie-Evolution

Versionierung und Evolution spielt auch im Bereich der Ontologieerstellung eine wichtige Rolle, da dies zumeist einen kollaborativen Prozess darstellt. Zwischen Schema-Evolution und Ontologie-Evolution liegen dabei viele Ähnlichkeiten vor [NK04], es herrschen aber auch wesentliche Unterschiede, wie die Verwendung von kontrollierten Vokabularen, Taxonomien und regelbasierten Wissensrepräsentationen [RB06]. Daraus resultieren unterschiedliche Änderungen, die während der Evolution vorgenommen werden können. Da Ontologien oft schemaartige konzeptionelle Metadaten und Instanzdaten enthalten, müssen die resultierenden Auswirkungen auf Meta- und Instanzdaten gemeinsam be-

achtet werden [RB06]. Des Weiteren kann eine Domänenontologie in unterschiedlichen Anwendungen wiederverwendet werden, woraus Abhängigkeiten zwischen verteilten Systemen resultieren [RB06]. Eine traditionelle Trennung von Versionierung und Evolution kann daher bei Ontologien nicht angewendet werden [NK04]. Resultierende Konzepte zur Unterstützung von Ontologie-Evolution werden beispielsweise in [Sto04; Noy+06] vorgestellt.

2.3.2.3 Schema-Evolution vs. Ontologie-Evolution

Djedidi und Aufaure [DA10] geben in ihrer Veröffentlichung einen Vergleich über die Evolution von objektorientierten Datenbankschemas und Ontologien. In der folgenden Tabelle 2.1 werden wesentliche Eigenschaften in Anlehnung an [DA10] gegenübergestellt. Eine vollständige Auflistung ist in [DA10] zu finden.

Tabelle 2.1: Vergleich von Schema-Evolution und Ontologie-Evolution (basierend auf [DA10])

Eigenschaft	**Datenbankschema**	**Ontologie**
Häufigkeit der Änderungen	Häufige Änderungsoperationen über gesamten Datenbanklebenszyklus	Häufigkeit von Änderungen besonders hoch, wenn Änderungsanforderungen durch Nutzer erfasst werden
Struktur	Spiegelt die Struktur von Daten und Code wider, berücksichtigen auch Objektverhalten (z.B. Methoden im Modell)	Spiegelt eine Domänenstruktur unter Nutzung von Konzepten, Beziehungen und Einschränkungen wider
Instanzen	Instanzen (Datenbankobjekte) befinden sich nicht auf dem gleichen Niveau wie Klassen	Terminologieebene (Klassen und Eigenschaften) und Aussagenebene (Instanzen) werden nicht abgegrenzt, Klassen und Instanzen können gemeinsam in Anfragen manipuliert werden
Wiederverwendung	Integration von einem Schema in ein anderes ist nicht möglich	Ontologien können vollständig oder teilweise wiederverwendet werden, Änderungen können auch die Hinzufügung oder Entfernung einer Ontologie sein

Tabelle 2.1: Vergleich von Schema-Evolution und Ontologie-Evolution (basierend auf [DA10]) *(Fortsetzung)*

Eigenschaft	Datenbankschema	Ontologie
Konsistenz	Durchführbare Änderungen müssen explizit definiert werden (auf dem Modell selbst)	Neben der Darstellung von Änderungen gemäß Ontologie ist Änderungssemantik essenziell (überprüfbare Bedingungen und Aktionen für Aufrechterhaltung der Konsistenz)
Konsistenzprüfung	Schemasemantik ist nicht ausreichend explizit, um automatisierte Schlussfolgerungen zu ziehen	Ontologiesemantik ist explizit und erlaubt die Anwendung von automatisierten Schlussfolgerungsmechanismen, um Inkonsistenzen zu detektieren
Propagierung von Änderungen	Änderungspropagierung auf Instanzen beschränkt	Änderungspropagierung auf alle Ontologie-abhängige Artefakte, wie Instanzen, Annotationen, Ontologien und Anwendungen

2.4 Revisionsverwaltung

Wie in den vorangehenden Abschnitten dargestellt, besitzen Modelle einen Lebenszyklus, der durch eine natürliche Evolution mit daraus resultierenden Änderungen gekennzeichnet ist. Auftretende Änderungen, die beispielsweise aufgrund von Wartung und Pflege der Modelle entstehen, müssen aus Gründen der Wartbarkeit und Nachvollziehbarkeit stets gespeichert werden. Soll im Fall von Fehlern oder bei einer parallelen Entwicklung automatisch auf einen funktionsfähigen Stand zurückgesprungen oder unabhängige Änderungen zusammengefügt werden können, ist die Anwendung von Revisionskontrollmechanismen sinnvoll. Insbesondere in verteilten und agilen Entwicklungsprozessen ist es hilfreich, wenn diese auch dazu befähigen, dass die Auswirkung von Änderungen auf verknüpfte Modelle automatisch erkannt werden, um stets ein konsistentes Gesamtgefüge an einzelnen Modellen zu gewährleisten. Versions- beziehungsweise Revisionsverwaltungssysteme stellen die Funktionalität bereit, die Evolution von Informationsmodellen zu dokumentieren und nachvollziehbar zu speichern. Beispielsweise kann hierdurch bei Defekten auf einen vorangegangen funktionsfähigen Stand zurückgesprungen werden.

2.4.1 Terminologie

Im Folgenden wird eine Einordnung der Begriffe Versions- beziehungsweise Revisionsverwaltung vorgenommen und auf weitere wichtige Begrifflichkeiten in diesem Zusammenhang eingegangen.

2.4.1.1 Revisionsverwaltung vs. Versionsverwaltung

In der Literatur wird der Begriff Versionsverwaltung im Wesentlichen für ein System verwendet, das Änderungen erfassen und verwalten kann. Die Erfassung und die Möglichkeit der Rücknahme von vorgenommenen Änderungen kommt vor allem bei der Entwicklung von komplexen Systemen zum Tragen, da in diesem Fall eine Koordination der beteiligten Entwickler über einen längeren Zeitraum erfolgen muss [Bae05]. Die Definition von Baerisch bezieht sich, wie die meisten in der Literatur vorhandenen, auf Softwaresysteme und die Versionierung von Source Code und anderen Dateien. Sie kann aber auch in dieser Arbeit verwendet werden, da sie sich allgemeingültig auf die beiden Hauptelemente von Versionsverwaltung bezieht. Die Erweiterung von Versionsverwaltung in dieser Arbeit betrifft die weitgehendere Nutzung von Versionierung für Informationsmodelle und abgeleitete Instanzdaten. Diese ist mit etablierten Versionsverwaltungssystemen wie git[7)] oder Apache Subversion (SVN)[8)] nicht möglich, da sie Änderungen auf einer zeilenbasierten Ebene betrachten und nicht auf einer inhaltlichen. Die Zeilenordnung spielt bei den in dieser Arbeit betrachteten Modellen keine Rolle. Diese kann sich ändern, ohne dass sich der Modellinhalt ändert.

Unabhängig vom verwendeten Versionsverwaltungssystem muss zwischen den Begriffen Version und Revision unterschieden werden. Eine Revision kennzeichnet jeweils einen eindeutigen Versionsstand des gesamten Repositories [Bud09; Fog05]. Dieser ist durch einen Identifier (Revisionsnummer) eindeutig bestimmt. Möglich sind beliebige Schemas, wie zum Beispiel eine fortlaufende Nummerierung oder ein Hash. Für die Speicherung der Revisionen wird zumeist eine Differenzspeicherung genutzt. Bei dieser werden nur die Unterschiede zwischen den aufeinander aufbauenden Revisionen gespeichert, was in einem verringerten Speicherplatzbedarf resultiert. Jedoch wird für die Wiederherstellung der vollständigen Revisionsinformation wiederum mehr Rechenaufwand benötigt. Eine Version bezieht sich auf eine bestimmte Veröffentlichung einer Revision, der eine eindeutige Versionsnummer zugeordnet ist [Bud09; Fog05]. Ein beispielhaftes Vergabeschema ist durch Semantic Versioning 2.0.0[9)] beschrieben. Dieses setzt sich aus *Major.Minor.Patch* zusammen. Neben den Begriffen Revision und Version wird in der Literatur auch teilweise der Begriff Variante verwendet. Varianten kennzeichnen dabei unterschiedliche Ausprägungen eines Produktes mit unterschiedlichem Funktionsumfang [DB07]. Eine Variante kann dabei beispielsweise von einer Version abgeleitet werden. Ein Beispiel hierfür ist Microsoft Windows in der Version zehn, das sowohl als Home-, wie auch Pro-Edition verfügbar ist. Eine interne Weiterentwicklung der Version zehn, die unter Umständen

[7)] https://git-scm.com/ (besucht am 29.11.2020)
[8)] https://subversion.apache.org/ (besucht am 29.11.2020)
[9)] http://semver.org/ (besucht am 29.11.2020)

nicht an den Kunden ausgeliefert wird, entspricht dabei einer Revision.

Die Begriffe Versionsverwaltung und Revisionsverwaltung werden in der Literatur synonym verwendet, wobei Versionsverwaltung der etablierte Begriff ist und Revisionsverwaltung den passenderen Begriff darstellt, da dieser feingranularer in Bezug auf die Verwaltungsebene der Änderungen ist. Im Folgenden wird daher die Bezeichnung Revisionsverwaltung verwendet.

2.4.1.2 Basisbegriffe der Revisionsverwaltung

An dieser Stelle erfolgt eine Definition von Basisbegrifflichkeiten der Revisionsverwaltung, die im weiteren Verlauf der Arbeit verwendet werden. Zur Veranschaulichung wird das in Abbildung 2.5 dargestellte Beispiel verwendet. Bei dem Repository handelt es sich um eine Datenbank, in der die gesamte Revisionshistorie vorgehalten wird. Eine Revision kennzeichnet, wie im vorangegangenen Abschnitt bereits eingeführt, einen eindeutigen Versionsstand des gesamten Repositories. Sie resultiert aus der Durchführung eines Commits, der durchzuführende Änderungen unter Angabe von Autor, Datum und einer zugehörigen Nachricht spezifiziert. Innerhalb des Repositories kann der Revisionsgraph aus mehreren Zweigen, sogenannten Branches, bestehen, die parallele Entwicklungen ermöglichen. Der Masterzweig gibt dabei den Hauptentwicklungszweig an. Die Zusammenführung von divergierten Entwicklungszweigen findet mittels Merges statt. Hierdurch wird die Revisionshistorie der beteiligten Zweige in eine neue Revision wieder zusammengeführt. Bei diesem Vorgehen können Konflikte auftreten, wenn auf den jeweiligen Branches Änderungen durchgeführt wurden, die zueinander im Widerspruch stehen. Schließlich gibt es die Möglichkeit einzelne Revisionen mit einem Tag zu versehen, wodurch diese eine besondere Kennzeichnung erhalten.

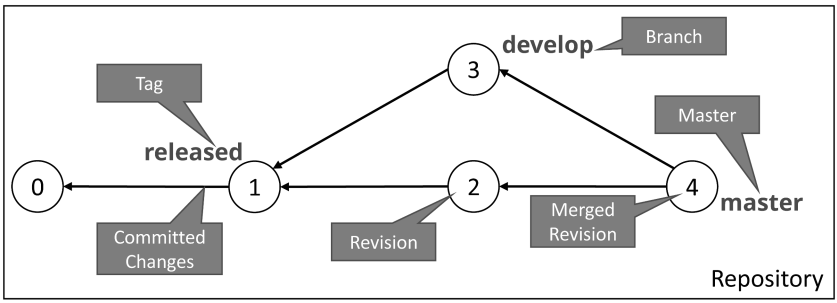


Abbildung 2.5: Begrifflichkeiten in der Revisionsverwaltung

2.4.1.3 Arten von Revisionsverwaltung

Revisionsverwaltungssysteme lassen sich nach [Sin11] in drei Arten beziehungsweise auch Generationen untergliedern. In der ersten Generation wird die Revisionierung vollständig

lokal durchgeführt. Die Möglichkeit, gleichzeitig Änderungen durch unterschiedliche Personen vorzunehmen, ist in diesem Fall nicht gegeben. Hierbei werden zudem meist nur einzelne Dateien revisioniert. Beispiele für lokale Revisionsverwaltung sind Revision Control System (RCS)[10], Source Code Control System (SCCS)[11], aber auch die Änderungsverfolgung in Microsoft Office. Die zweite Generation der Revisionsverwaltung verwendet einen zentralisierten Ansatz. Es handelt sich dabei um eine Client-Server-Architektur. Die Revisionsgeschichte wird im Repository auf dem Server vorgehalten, wodurch ein Zugriff über ein Netzwerk möglich ist. Bei nebenläufigen Änderungen von unterschiedlichen Personen müssen aber die bestehenden Änderungen zuerst in die eigene Arbeit gemerged werden, bevor ein Commit der eigenen Änderungen möglich ist. Beispiele für entsprechende Systeme sind SVN oder Concurrent Versions System (CVS)[12]. Generation drei bezeichnet die verteilte Revisionsverwaltung. In diesen Systemen existiert kein zentrales Repository und jede Person besitzt ein lokales Repository mit der gesamten Historie. Nebenläufige Änderungen können dementsprechend im jeweiligen lokalen Repository commited werden und müssen erst beim Abgleich zusammengeführt werden. Implementierungen für solche verteilten Systeme sind beispielsweise git, Bazaar[13] oder Mercurial[14]. Der Entwicklungsprozess über die Phasen hinweg zeigt eine stetige Evolution der Revisionsverwaltung zu mehr Nebenläufigkeit und der damit verbunden Möglichkeit von gleichzeitigen Änderungen der gleichen Datengrundlage.

2.4.1.4 Synchronisation und Replikation

Mutschler und Specht [MS04] definieren Replikation als die Einführung einer Redundanz von Daten. Diese Redundanz wird auch bei der Verwendung eines zentralen oder verteilten Revisionsverwaltunssystems hergestellt. Hierdurch ist eine Bearbeitung der Daten auch in Offline-Phasen möglich, in denen keine Verbindung zu einem Abgleichpunkt besteht. Änderungen werden dementsprechend an den replizierten Daten vorgenommen und müssen in einer folgenden Online-Phase synchronisiert werden, um die Änderungen auch anderen Teilnehmern in der Replikationsumgebung zur Verfügung stellen zu können [MS04].

Bei der Synchronisation werden nach Mutschler und Specht [MS04] redundant vorhandene Daten, an denen unterschiedliche Änderungen vorgenommen wurden, abgeglichen. Dieser Abgleich kann entweder von einer Replikationssenke auf eine Replikationsquelle (Reintegration) oder in umgekehrter Richtung (Rückübertragung) erfolgen. Hierbei wird von einem unidirektionalen Synchronisierungsprozess gesprochen. Dem gegenüber wird die direkte aufeinanderfolgende Ausführung von beiden Arten auch als bidirektionaler Synchronisationsprozess bezeichnet [MS04].

[10] https://www.gnu.org/software/rcs/ (besucht am 29.11.2020)
[11] http://sccs.sourceforge.net/ (besucht am 29.11.2020)
[12] https://savannah.nongnu.org/projects/cvs (besucht am 29.11.2020)
[13] http://bazaar.canonical.com/en/ (besucht am 29.11.2020)
[14] https://www.mercurial-scm.org/ (besucht am 29.11.2020)

2.4.1.5 Verfahren zur Konsistenzerhaltung

Innerhalb einer Replikationsumgebung muss Konsistenz zwischen den einzelnen Repliken in dieser Umgebung gewahrt werden. Die hierfür zur Verfügung stehenden Verfahren werden in optimistische und pessimistische Verfahren kategorisiert [MS04] und nachfolgend anhand von [MS04] vorgestellt. Pessimistische Verfahren (Lock-Modify-Write/Unlock) verwenden Sperren, um Inkonsistenzen auszuschließen. Hierbei werden die zu bearbeitenden Bereiche vor der Durchführung der Änderungen gesperrt und müssen nach der Bearbeitung wieder freigegeben werden. Optimistische Verfahren (Copy-Modify-Merge) erlauben hingegen die gleichzeitige Änderung durch mehrere Benutzer. Dies bedingt eine nachträgliche Zusammenführung der Divergenzen. Die entsprechenden Synchronisierungsverfahren werden unterteilt in konventionelle, die mittels einfacher Sperrverfahren realisiert werden, in zeitstempelbasierte, die zeitbehaftete Information nutzen, um eine Serialisierbarkeitsreihenfolge abzuleiten und in semantische Synchronisierungsverfahren, die spezielles Anwendungswissen einbeziehen.

2.4.2 Erweiterte Revisionskontrolle für Modelldaten

Neben den etablierten Revisionsverwaltungssystemen wie git und SVN werden auch neue Ansätze für die Revisionierung von Modelldaten entwickelt, die einen semantischen Ansatz verfolgen. Beispiele hierfür sind unter anderem im Semantic Web zu finden, wie SemVersion [VG06], R&W base [Van+13] oder R43ples [GHU14; GHU16].

Bestehende Systeme werden dabei meist auf technische oder funktionaler Ebene miteinander verglichen. Zum Beispiel analysieren Ekaputra *et al.* [Eka+15] die Eigenschaften verschiedener Systeme in Bezug auf die Unterstützung des Wissensänderungsmanagements in einer multidisziplinären Entwicklungsumgebung. Frommhold *et al.* [Fro+16] führten ebenso einen Funktionsvergleich durch, um notwendige Anpassungen für den eigenen Ansatz abzuleiten. Ferner haben Canova *et al.* [Can+15] vorhandene Lösungen wie R&Wbase [Van+13] und R43ples [GHU14; GHU16] analysiert, um erforderliche Schritte für die Revisionskontrolle kollaborativer offener Daten abzuleiten. Da sich die formale Beschreibung zwischen den Ansätzen erheblich unterscheidet, ist die Wiederverwendung von Komponenten oder Eigenschaften schwierig, da Interpretationsspielräume bestehen. In vielen der Fälle ist das Ergebnis der durchgeführten Analyse von bestehenden Systemen ein neues System, das die fehlenden Eigenschaften bereitstellt und teilweise auf den analysierten Systemen basiert. Neben der formalen Beschreibung ist ein weiterer wichtiger Aspekt die Beschreibung des Revisionsverlaufs. In den meisten Fällen sind diese Informationen nicht so transparent zugänglich, wie es beispielsweise von Klein und Fensel [KF01] gefordert wird. Außerdem müssen Interaktionsmöglichkeiten festgelegt werden, die es dem Benutzer ermöglichen, mit dem System zu interagieren.

Tabelle 2.2 stellt einen Vergleich von bestehenden Ansätzen hinsichtlich der bereitgestellten formalen Beschreibung (*Formal*), der Verwendung semantischer Beschreibungen für Revisionsinformation (*Semantik*) und der angebotenen Interaktionsmöglichkeiten (*Interaktion*) dar. Die analysierten Eigenschaften können vollständig (✓) oder nicht erfüllt (✗) sein. Falls eine Eigenschaft nur teilweise umgesetzt ist oder aufgrund einer

fehlenden Beschreibung der Funktionsumfang nicht abgeschätzt werden kann, dann ist die entsprechende Wertung in der Tabelle in Klammern angegeben. Um eine Eigenschaft vollständig zu erfüllen, müssen die bereitgestellten Beschreibungen grundlegende Revisionskontrollfunktionen, wie zum Beispiel Revisionsverlauf, Verzweigungen und Tags, und erweiterte Revisionskontrollfunktionen, wie die Zusammenführung von divergierten Entwicklungszweigen, enthalten.

Tabelle 2.2: Vergleich von bestehenden Ansätzen in Bezug auf die bereitgestellte formale Beschreibung

Ansatz	Formal	Semantik	Interaktion
Hauptmann *et al.* [HBW15]	✗	(✗)	✓
LUCID Endp. [Tra+15; Fro+16]	✗	✓	(✗)
Quit Store [ARM16; AM17]	✓	✗	✓
R&Wbase [Van+13]	✗	✓	✓
R43ples [GHU14; GHU16]	(✓)	✓	✓

Hauptmann *et al.* [HBW15] stellen die Interaktion mit dem Revisionskontrollsystem über SPARQL Protocol And RDF Query Language (SPARQL)-Anfragen bereit. Die zugehörige Revisionsinformation sollte ebenso über SPARQL zugänglich sein, die zur Beschreibung genutzte Ontologie wird jedoch nicht bereitgestellt. Des Weiteren existieren keine formalen Definitionen der Revisionsfunktionalitäten. Der LUCID-Endpunkt [Tra+15; Fro+16] verwendet eine Ontologie, um die grundlegende Revisionsinformation zu beschreiben. Mathematische Formalismen oder Interaktionsmechanismen, um auf eine bestimmte Revision zuzugreifen, werden jedoch nicht dargelegt. Der Quit Store [ARM16; AM17] ist auf der Revisionsgraphebene bezüglich der grundlegenden und erweiterten Revisionskontrollfunktionen formal definiert, der Revisionsgraph selbst jedoch nicht. Die Revisionsinformation ist des Weiteren nicht semantisch beschrieben bereitgestellt, da git als zugrunde liegendes Revisionskontrollsystem verwendet wird. Kollaborative Interaktionen werden mit Hilfe von git-Befehlen realisiert, und Aktualisierungen können durch SPARQL-Anfragen durchgeführt werden. R&Wbase [Van+13] verwendet eine Ontologie, um dem Benutzer die Revisionsinformation bereitzustellen. Die Abfragemechanismen, um auf Revisionen zuzugreifen, werden ebenfalls erläutert. Es existiert jedoch keine formale Definition des gesamten Systems. R43ples [GHU14; GHU16] enthält nur formale Definitionen in Bezug auf die automatisierte Anpassung von SPARQL-Anfragen, um die Performance des Systems zu erhöhen. Die Revisionsinformation wird mithilfe einer Ontologie beschrieben und die Interaktion mit dem System wird über erweiterte SPARQL-Anfragen realisiert. Es werden nur grundlegende Revisionskontrollfunktionen bereitgestellt.

2.5 Konsistenz

Während der Evolution von Modellen muss stets ein konsistenter Stand der Verbindungen zwischen den Modellen, aber auch innerhalb einer Replikationsumgebung, hergestellt

werden. Da der Begriff Konsistenz in unterschiedlichen Wissenschaftsbereichen verschiedene Verwendung beziehungsweise auch Interpretation erfährt, wird in diesem Abschnitt ein grundlegender Überblick der Begrifflichkeit und deren Verwendung gegeben.

2.5.1 Terminologie

Auf Basis von unterschiedlichen Definitionen des Konsistenzbegriffes wird eine Definition für die weitere Arbeit abgeleitet. Des Weiteren wird eine Klassifikationsmöglichkeit von Modellkonsistenz vorgestellt, was eine Unterscheidung von unterschiedlichen Typen von Konsistenz ermöglicht.

2.5.1.1 Konsistenz

Dem Begriff Konsistenz werden im Duden drei verschiedene Bedeutungen zugeordnet. So kennzeichnet dieser in der Fachsprache oft „Grad und Art des Zusammenhalts eines Stoffes“[Dud]. Im bildungssprachlichen Gebrauch wird er in Form von „konsistente Beschaffenheit“[Dud] verwendet. Schließlich erfolgt im Logikbereich die Verwendung vor allem als „strenger gedanklicher Zusammenhang“[Dud]. Für diese Arbeit sind die beiden letzteren Bedeutungen von Relevanz, die wiederum in unterschiedlichen Bereichen Verwendung finden. Ausprägungen des Konsistenzbegriffs lassen sich beispielsweise im Gebiet von Datenbanken vor allem in Bezug auf transaktionale Konsistenz [Fre06], im Zusammenhang mit Modellen [LMT09; SZ01], in der Mathematik (auch unter dem Begriff Widerspruchsfreiheit) [Glo06], in der Softwareentwicklung [ISO10], aber auch in der Dialoggestaltung im Bereich der Erwartungskonformität [DIN06] finden.

Aus den unterschiedlichen Gebieten heraus ergeben sich Definitionen von Konsistenz. Hofstadter [Hof79] beispielsweise definierte diese für formale Systeme wie folgt:

> „Widerspruchsfreiheit [...] [ist keine] Eigenschaft eines formalen Systems als [...] [solches], sondern [ist] von der dafür vorgeschlagenen Interpretation abhängig [...].“[Hof06, S. 103] [15)]

Widerspruchsfreiheit ist demnach nach Hofstadter [Hof06] von der für das formale System vorgeschlagenen Interpretation abhängig und nicht eine Eigenschaft des Systems an sich. Das bedeutet, dass bei Widerspruchsfreiheit gelten muss, dass jeder Satz des formalen Systems bei seiner Interpretation eine wahre Aussage ergibt [Hof06].

Im Bereich der Modellkonsistenz hat beispielsweise Stevens [Ste08] folgende informelle Definition von Konsistenz bei der Anwendung von bidirektionalen Transformationen gegeben:

> Zwei Modelle sind konsistent, wenn es für alle ihre Stakeholder akzeptabel ist, mit diesen Modellen fortzufahren, ohne eines zu ändern.[16)] (nach [Ste08, S. 411])

[15)] Übersetzung aus dem Englischen: „[...] consistency is not a property of a formal system per se, but depends on the interpretation which is proposed for it.“[Hof79, S. 102]

[16)] Übersetzung des Autors aus dem Englischen: „[...] two models are consistent if it is acceptable to all their stakeholders to proceed with these models, without modifying either.“[Ste08, S. 411]

Weiterhin wird in Bezug auf Replikation und Synchronisation auch von Replikationskorrektheit gesprochen. Mutschler und Specht [MS04] geben hierfür die nachfolgende Definition, die den Konsistenzbegriff wiederum aufgreift.

> „Replikationskorrektheit heisst [sic!], dass alle redundanten Kopien stets konsistent im Sinne der jeweils definierten Korrektheitskriterien sind. [...]“[MS04, S. 86]

In verteilten Systemen allgemein kommt nach Schmidt [Sch16] Konsistenz vor allem in Bezug auf die Replikation zum Tragen. Hierbei verfolgt die Replikation zwei Ziele. Zum einen kann die Verlässlichkeit von Diensten beziehungsweise die Verfügbarkeit von Daten erhöht werden, zum anderen ist auch eine Leistungsfähigkeitssteigerung möglich, wenn auf ein Datum zugegriffen wird. Die replizierten Daten müssen konsistent zueinander gehalten werden, um daraus resultierende Fehler zu vermeiden. Hierfür kommen Konsistenzmodelle zum Einsatz, die als eine Art Vertrag gesehen werden. Dieser Vertrag existiert zwischen dem Datenspeicher und den Prozessoren, die Zugriff auf diesen haben. Je nach Sicht wird dabei in Daten-zentrierte und Client-zentrierte Konsistenzmodelle unterschieden. Davon abgeleitet existieren weitere Konsistenzarten, wie zum Beispiel strikte/atomare oder sequentielle Konsistenz.

Die hier aufgeführten Definitionen lassen den Schluss zu, dass Konsistenz jeweils abhängig von der Domäne der Anwendung und dem konkreten Anwendungsfall ist. Konsistenz setzt demnach voraus, dass innerhalb der jeweiligen Umgebung, die konsistent gehalten werden soll, eine Vereinbarung der beteiligten Parteien über den Nachweis von Konsistenz herrscht. Dieser Nachweis lässt sich zumeist in einer Art Regelsatz festlegen, der erfüllt werden muss. Demnach wird für die weitere Arbeit die nachfolgende Definition von Konsistenz angenommen.

> ***Definition Konsistenz:***
> *Konsistenz innerhalb einer Umgebung tritt dann ein, wenn Vereinbarungen der beteiligten Parteien über ihren Nachweis (zum Beispiel in Form von Regelsätzen) erfüllt werden.*

Bei der Nichteinhaltung beziehungsweise der Verletzung der vereinbarten Regeln treten Inkonsistenzen auf. In [LMT09] werden nach [Huz+04] zwei Hauptgründe für das Auftreten von Inkonsistenzen bei Modellen genannt. Einerseits werden Systeme durch unterschiedliche Sichten beschrieben. Diese stellen jeweils bestimmte Details bereit und die Gesamtheit der Einzelmodelle bildet dann das Gesamtsystem. Zwischen den einzelnen Sichten besteht dabei auch die Möglichkeit der Überlappung. Andererseits werden Systeme über mehrere Phasen beziehungsweise in mehreren Iterationen entwickelt. Jede erzeugt dabei eine verfeinerte Beschreibung des Systems. Überdies wird ein weiterer Grund aufgeführt, der aus der verteilten Entwicklung mit potenziell mehreren Entwicklern, die auch geographisch verteilt sein können, resultiert. Hierdurch kann es zu unterschiedlichen Interpretationen der Anforderungen oder der UML Notation an sich kommen.

2.5.1.2 Klassifikation von Modellkonsistenz

Während des Modelllebenszyklus und der damit einhergehenden Evolution können Änderungen auf unterschiedlichen Ebenen und mit unterschiedlichen Auswirkungen auftreten. Für die weitere Arbeit wird daher an dieser Stelle die Unterteilung der Modellkonsistenz nach Lucas *et al.* [LMT09] vorgestellt. Hierbei greifen Lucas *et al.* auf bestehende Definitionen von Engels *et al.* [Eng+01] und Huzar *et al.* [Huz+04] aus der UML-Domäne zurück und unterteilen die Modellkonsistenz in die nachfolgenden vier Typen.

Horizontal Horizontale Konsistenz wird nach [LMT09; Eng+01; Huz+04] auch als intra-Modellkonsistenz bezeichnet. Es handelt sich hierbei um die Konsistenz zwischen Modellen, die auf dem gleichen Level der Modellabstraktion liegen. Daraus resultierende Konsistenzprobleme entstehen beispielsweise, wenn eine Spezifikation aus unterschiedlichen Teilen besteht, die jeweils unterschiedliche Aspekte fokussieren, wie die Beschreibung der statischen und der dynamischen Sicht auf die modellierte Domäne. Die jeweils modellierten Sichten müssen zueinander konsistent und nicht widersprüchlich sein, um zum Beispiel später eine korrekte Implementierung ableiten zu können.

Vertikal Im Gegensatz zur horizontalen Konsistenz bezieht sich die vertikale oder auch inter-Modellkonsistenz nach [LMT09; Eng+01; Huz+04] auf die Konsistenz zwischen Modellen, die auf unterschiedlichen Abstraktionsleveln aufbauen. Innerhalb der Entwicklung unterliegen die Modelle einem stetigen Verfeinerungsprozess. Während dieses Prozesses sollen die jeweils erzeugten Modelle aber mit dem Modell auf der höheren und mehr abstrakteren Ebene vertikal konsistent sein. Ein Beispiel hierfür ist die Konsistenz zwischen einem Analyse- und einem Designmodell.

Syntaktisch Nach [LMT09; Eng+01] muss zur Erreichung einer syntaktischen Konsistenz gewährleistet sein, dass ein Modell mit der abstrakten Syntax übereinstimmt, die durch das Metamodell definiert wird. Das bedeutet, dass das Gesamtmodell wohlgeformt sein muss. Als Vergleichsbeispiel kann hier die Syntax in Programmiersprachen herangezogen werden.

Semantisch Semantische Konsistenz ist nach [LMT09; Eng+01] sehr stark von der zugrunde liegenden Semantik des Modells und dem Entwicklungsprozess abhängig, setzt aber immer syntaktische Konsistenz voraus. Das Modellverhalten der zueinander semantisch konsistenten Modelle muss daher jeweils semantisch kompatibel sein, was einer Übereinstimmung auf der Bedeutungsebene entspricht. Beispielsweise muss demnach für ein horizontales Konsistenzproblem gelten, dass die Modelle der unterschiedlichen Sichten in Bezug auf die Aspekte des Systems, die in beiden Submodellen spezifiziert werden, semantisch kompatibel sind. Ähnliches gilt für vertikale Konsistenzprobleme, wobei hier das verfeinerte Modell semantisch kompatibel zum verfeinernden Modell sein muss.

2.5.2 CAP-Theorem

Wie in den vorangegangenen Abschnitten dargestellt, spielt Konsistenz eine wichtige Rolle bei der Evolution von Systemen und den zugrunde liegenden Modellen. Insbesondere in verteilten Systemen sind jedoch weitere Eigenschaften von Bedeutung, auf die neben der Konsistenz *(engl. „Consistency“)*, innerhalb des Consistency, Availability, Partition Tolerance (CAP)-Theorems Bezug genommen wird. Das im Jahr 2000 von Brewer vorgestellte CAP-Theorem (auch als Brewer-Theorem bezeichnet) sagt aus, dass ein verteiltes System immer nur zwei der drei Eigenschaften (Consistency, Availability, Partition Tolerance) erfüllen kann [Bre00]. Durch Gilbert und Lynch [GL02] erfolgte im Jahr 2002 der Beweis des Theorems.

Nach Kolb [Kol14] werden im Folgenden die einzelnen Eigenschaften dargestellt. Im Rahmen des CAP-Theorems sagt Consistency (Konsistenz) aus, dass innerhalb des verteilten Systems alle Knoten zu jedem Zeitpunkt die gleiche Sicht auf die Daten haben. Availability (Verfügbarkeit) ist dann gegeben, wenn das System alle Lese- und Schreibanfragen beantwortet. Die Verfügbarkeit von funktionsfähigen Knoten wird dabei durch den Ausfall von anderen Knoten nicht beeinflusst. Die Eigenschaft der Partition Tolerance (Partitionstoleranz) gibt an, dass die Funktionsfähigkeit des Systems trotz Kommunikationsunterbrechungen zwischen Knoten und damit einhergehenden Verlusten von Nachrichten gewahrt bleibt.

Aus den gegebenen Definitionen lassen sich CAP-Fälle ableiten, die jeweils nur zwei der drei Eigenschaften erfüllen und damit entweder keine Partitionstoleranz (CA-System), keine Verfügbarkeit (CP-System) oder keine Konsistenz (AP-System) bereitstellen [Kol14]. Insbesondere bei CP- und AP-Systemen ist es jedoch auch möglich, Mischformen zu etablieren, die je nach Situation zwischen der Verfügbarkeit und der Konsistenz auswählen. Dies hat Brewer [Bre00] bereits legitimiert, da er beim Vergleich der beiden Hauptansätze Atomicity, Consistency, Isolation, Durability (ACID) und Basically Available, Soft state, Eventual consistency (BASE) diese als ein Spektrum bezeichnet hat. ACID stellt dabei die Eigenschaften dar, die eine strenge Konsistenz bei der Ausführung von Transaktionen ermöglicht [HR83; Bre00; Vos09]. Dies hat aber auch zur Folge, dass die Evolution, beispielsweise von Datenbankschemas, erschwert wird. Im Gegensatz hierzu operiert der BASE-Ansatz als ein sehr optimistischer Ansatz. Diesem wird der Grundsatz der Eventual Consistency zugrunde gelegt, in dem die Konsistenz als ein Zustand zu verstehen ist, der zu irgendeinem Zeitpunkt eingenommen wird. Inkonsistente Zustände werden für die Erreichung von hoher Verfügbarkeit in Kauf genommen [SK09; Bre00].

3 Analyse

In diesem Abschnitt werden ausgehend von einer Literaturrecherche Anforderungen an ein System zur Unterstützung der Evolution von Informations- und Datenmodellen gestellt. Die aufgenommenen Anforderungen werden durch die Analyse der Anwendungsfälle erweitert beziehungsweise bestätigt. Auf dieser Basis werden bestehende Ansätze gegen die aufgenommenen Anforderungen geprüft. Schließlich werden für die Umsetzung notwendige Technologien auf deren Eignung hin geprüft und ausgewählt. Als letztes erfolgt eine Zusammenfassung der Analyseergebnisse.

3.1 Anforderungsanalyse

Ausgangspunkt für die durchgeführte Literaturrecherche ist ein von Bahill und Botta veröffentlichter Journalbeitrag [BB08]. In diesem werden fundamentale Prinzipien für ein gutes Systemdesign aus unterschiedlichen Domänen abgeleitet, wozu unter anderem Hardware-, Software-, System- und Testdesign zählen. Den Prinzipien werden in [BB08] unterschiedliche Aspekte gegenübergestellt und gekennzeichnet, welches Prinzip auf welchen Aspekt einen Einfluss hat.

In den folgenden Abschnitten werden zuerst die Prinzipien aus [BB08], die einen Einfluss auf den Aspekt der Evolvability besitzen, näher beleuchtet, da in dieser Arbeit das Zusammenspiel zwischen Evolutionsmechanismen und Revisionskontrolle untersucht wird. Anschließend erfolgt eine Betrachtung von weiteren Anforderungen aus einer technologischen Sicht heraus. Abschließend werden die Anforderungen mittels der Anwendungsfälle erweitert beziehungsweise bestätigt und tabellarisch zusammengefasst.

3.1.1 Prinzipien mit Einfluss auf Evolvability

In den nachfolgenden Unterabschnitten wird jeweils ein Prinzip aus [BB08] eingeführt, das einen Einfluss auf den Aspekt der Evolvability besitzt. Hierbei wird jeweils eine kurze Beschreibung anhand von [BB08] vorgenommen und nachfolgend werden mögliche Beiträge von Revisionskontrolle zur Erreichung des aufgeführten Prinzips erläutert. Die Abschnittsüberschriften sind des Weiteren mit einem Tupel gekennzeichnet. Das erste Element beschreibt hierbei den Einfluss des Prinzips auf die Evolvability (X steht für einen bedeutenden Einfluss, x steht für einen weniger bedeutsamen aber immer noch sehr großen Einfluss). Das zweite Element des Tupels gibt an, ob aus der Nutzung des Prinzips zusätzliche Kosten entstehen. Dies wird durch ein $-Zeichen gekennzeichnet. Wenn keine zusätzlichen Kosten entstehen, so wird dies durch einen waagerechten Strich angegeben.

3.1.1.1 P1 - Entwicklung von stabilen Zwischenergebnissen (X,-)

Prinzipbeschreibung nach [BB08] Größere Änderungen sollen stets so erstellt werden, dass diese wiederum aus einer Serie von kleineren gebildet werden. Diese kleineren Änderungen sollen selbst wiederum so stabil sein, dass die Entwicklung an vordefinierten Punkten gestoppt werden kann und trotzdem etwas Nützliches weiterbesteht.

Beitrag Revisionskontrolle Durch die Verwendung von Revisionskontrolle wird die notwendige Infrastruktur bereitgestellt, die es ermöglicht, die Nachvollziehbarkeit der Änderungen zu gewährleisten [Vog+15b]. Einzeländerungen können zu größeren Sinneinheiten gruppiert werden. Des Weiteren besteht die Möglichkeit der Verwendung von Rücksprüngen auf vorherige Entwicklungen und die Verwendung von Entwicklungszweigen. Dies geht einher mit der Notwendigkeit für die Zusammenführung von divergierten Entwicklungszweigen. Die in der Prinzipbeschreibung angeführten kleinen Änderungen können dementsprechend als Revisionen und die aggregierten Änderungen als eine Version bezeichnet werden [Bud09; Fog05]. Außerdem können Tags erstellt werden, die den jeweiligen Entwicklungsstand kennzeichnen.

3.1.1.2 P2 - Nutzung von evolutionärer Entwicklung (X,$)

Prinzipbeschreibung nach [BB08] Der Start für die Evolution sollte immer ein nutzbares System sein. Erst im Folgenden können dann weitere Anforderungen hinzugefügt und mit entsprechenden zusätzlichen finanziellen Mitteln umgesetzt werden. Hierdurch kann dann wiederum ein komplexeres und nutzbares System erreicht werden.

Beitrag Revisionskontrolle Analog zu Prinzip P1 kann durch Revisionskontrolle wiederum die notwendige Infrastruktur bereitgestellt werden, um die Nachvollziehbarkeit zu gewährleisten. Weiterhin können bereits durchgeführte Releases vorgehalten werden und von diesen aus können wiederum Weiterentwicklungen stattfinden. Dabei wird der aktuelle funktionsfähige Stand nicht gefährdet, da Entwicklungszweige und Tags benutzt werden können.

3.1.1.3 P3 - Verständnis des Unternehmens (X,$)

Prinzipbeschreibung nach [BB08] Wichtig ist hierbei, ein Verständnis zu erlangen, wie sich das zu designende System in das Unternehmen eingliedert, wozu Frameworks benutzt werden können. Diese dienen zum einen zur Organisation und zum anderen kann durch deren Nutzung auch die Vollständigkeit von existenten Modellen beurteilt werden. Durch die Nutzung von Frameworks kann außerdem definiert werden, welche Aspekte aus welcher Perspektive betrachtet werden sollen.

Beitrag Revisionskontrolle Revisionskontrolle erfüllt in diesem Zusammenhang vorwiegend einen Dokumentationsaspekt. Dadurch ist stets eine Nachvollziehbarkeit gegeben.

Aus dieser kann wiederum abgeleitet werden, welche Entscheidung welche technische Umsetzung nach sich gezogen hat.

3.1.1.4 P4 - Bereitstellung von überprüfbaren Zuständen (x,$)

Prinzipbeschreibung nach [BB08] Für die Überprüfung von Systemäquivalenzen wird das Zustandsverhalten für dynamische Systeme benötigt. Ein reines Ein-/Ausgangsverhalten ist dabei nicht ausreichend. Im Idealfall ist ein vollständiger Systemstatus verfügbar, mindestens jedoch Resetzustände und Wiederherstellungspunkte. Durch die Bereitstellung von Zuständen und dem damit einhergehenden Wissen über das Verhalten können bestehende Systeme wiederverwendet, Systeme geupgradet, kommerzielle Standardprodukte integriert, Feldausfälle repliziert und sich weiterentwickelnde Systeme verifiziert werden.

Beitrag Revisionskontrolle Durch eine Revisionierung der Zustandsinhalte kann bei der Durchführung von entsprechenden Up- beziehungsweise Downgrades gegen die revisionierten Zustände geprüft werden.

3.1.1.5 P5 - Nutzung von offenen Standards (x,-)

Prinzipbeschreibung nach [BB08] Es sollten öffentlich verfügbare Spezifikationen verwendet werden. Diese können von allen eingesehen und implementiert werden. Beispiele hierfür sind unter anderem UML und Universal Serial Bus (USB). Im Gegensatz hierzu operieren proprietäre Standards, die nur durch eine einzelne Entität kontrolliert werden.

Beitrag Revisionskontrolle In diesem Zusammenhang sollten auch bei der Revisionskontrolle offene und interoperable Technologien beziehungsweise Standards eingesetzt werden. Die durch die Verwendung von offenen Standards bereitgestellte Information kann ebenso im Revisionkontrollsystem Verwendung finden, um beispielsweise die Revisionsinformation zusätzlich semantisch anzureichern [GHU14].

3.1.1.6 P6 - Identifizierung von Dingen, die sich wahrscheinlich ändern (X,-)

Prinzipbeschreibung nach [BB08] Es sollte stets zwischen Aspekten unterschieden werden, die eine hohe Wahrscheinlichkeit für Änderungen aufweisen und denen, die eine hohe Wahrscheinlichkeit für die Beständigkeit besitzen. Bei den Aspekten, die sich wahrscheinlich ändern werden, sollte zum einen eine zusätzliche Anstrengung in die Schnittstellenentwicklung fließen und zum anderen sollen Möglichkeiten vorgesehen werden, um korrespondierende Änderungen aufnehmen zu können.

Beitrag Revisionskontrolle Aufgrund der durch die Revisionskontrolle bereitgestellten Änderungshistorie können Aspekte abgeleitet werden, die einem hohen Änderungspotential unterliegen. Des Weiteren kann bei der Aufnahme von Änderungen unterstützt

werden. Hierfür sind jedoch weitreichendere Mechanismen vorzusehen, die beispielsweise Co-Evolutionen oder semantische Konflikterkennung und -behebung unterstützten [Ruh+14].

3.1.1.7 P7 - Design für Evolvability (X,$)

Prinzipbeschreibung nach [BB08] Um mit Änderungen während des Lebenszyklus umgehen zu können, wird eine hohe Flexibilität und Adaptivität benötigt. Auf Änderungen kann unter anderem durch die Rekonfiguration von existenten Entitäten, der Vergrößerung von Entitäten oder der Hinzufügung von neuen Entitäten reagiert werden.

Beitrag Revisionskontrolle Änderungen können durch das Revisionskontrollsystem nachvollziehbar gespeichert werden. Auf dieser Basis besteht die Möglichkeit, semantische Anreicherungen vorzunehmen beziehungsweise semantische Änderungen zu erkennen [Keh15], um damit Co-Evolutionen [Ruh+14] oder semantische Konflikterkennung und -behebung umzusetzen, was wiederum die Evolvability des Systems erhöht.

3.1.2 Technologische Sicht

Aus der in Abschnitt 3.1.1 durchgeführten Gegenüberstellung der Prinzipien aus [BB08] und den möglichen Beiträgen von Revisionskontrolle ergibt sich bereits, dass etablierte Revisionskontrollsysteme nicht alles leisten können, was zur Erreichung der Prinzipien notwendig ist. Speziell für die Ontologieevolution wird durch Noy und Klein [NK04] ausgesagt, dass eine Trennung von Revisionierung und Evolution nicht anwendbar ist und es sich hierbei vielmehr um das Management von Ontologienänderungen und deren Auswirkungen handelt. Dementsprechend werden im Folgenden Anforderungen zum Übergang von getrennten Systemen für Revisionskontrolle und Evolution hin zu einem integrierten *Revision Management System* aufgenommen. Hierfür wird die Literaturrecherche vor allem in Bezug auf technologische Anforderungen vertieft, die ein solches System erfüllen muss. Ausgangspunkt sind die im Abschnitt 2 bereits verwendeten Quellen, die den Stand der Technik widerspiegeln.

3.1.2.1 Nutzungskontext

Für die Entwicklung eines Revision Management Systems muss als erstes der Nutzungskontext untersucht werden, auf dem alle weiteren Anforderungen aufbauen. Im Zentrum steht hierbei der Nutzer des Systems, der unterstützt werden soll und nur noch an den Stellen manuell eingreifen muss, an denen dies zwingend notwendig ist. Die Verwaltung von Änderungen soll für den Nutzer entsprechend vereinfacht werden, wobei das System sowohl von Anfängern als auch Experten gleichermaßen benutzbar sein soll [Sto04].

Levendovszky *et al.* definieren weiterhin sechs verschiedene Rollen, die am Evolutionsprozess eines Modells beteiligt sind. Hierzu zählen Model Designer, Model Evolver, Language Evolver, Requirements Specifier und Requirements Evolver [Lev+10]. Vor allem in größeren Projekten werden diese Rollen auf mehrere Personen aufgeteilt.

Da es sich hierbei um eine kollaborative Umgebung handelt, in der mehrere Personen beteiligt sein können, müssen auch entsprechende Mechanismen vorhanden sein, um Zugriffe einschränken zu können [Noy+06]. Dafür müssen Mechanismen zur Rechtevergabe für bestimmte Nutzer beziehungsweise Nutzergruppen oder Nutzerrollen etabliert und deren Einhaltung überprüft werden.

Wie bereits aus den Rollen von Levendovszky ersichtlich, müssen in den Unternehmen bestimmte Prozesse eingehalten werden. Hierzu gehört auch die Freigabe von Änderungen durch zum Beispiel einen Kurator [Noy+06]. Dieser ist für die Durchführung einer Qualitätskontrolle verantwortlich, bevor beispielsweise eine neue Version ausgeliefert wird. Nach Noy *et al.* [Noy+06] sind hierfür spezielle Sichten auf die Änderungen notwendig, die es erlauben, einzelne oder Gruppen von Änderungen anzunehmen oder abzulehnen. Wichtig sind hierbei Sichten auf konfliktbehaftete Änderungen, den Kontext (Person, Zeit, Änderungen) und zugehörige Filtermöglichkeiten. Der Änderungsprozess soll des Weiteren unterbrochen und zu einem späteren Zeitpunkt ohne Verluste von bisher durchgeführten Reviews fortgesetzt werden können.

Anforderungen inklusive Kurzbeschreibung

- Selbstbeschreibungsfähigkeit des Systems

  Die Nutzung des Systems soll unabhängig von Vorkenntnissen von unterschiedlichen Nutzern genutzt werden können.

- Rollenmanagement für Nutzer

  Das System unterstützt die Verwaltung und Nutzung von unterschiedlichen Rollen, die am Evolutionsprozess beteiligt sind.

- Zugriffsmanagement mit Rechtevergabe

  Beschränkung von Zugriffsrechten und Verwaltung von Rechten für Nutzer, Nutzergruppen und Nutzerrollen.

- Umsetzung von Freigabeprozessen

  Möglichkeit der Etablierung von Freigabeprozessen mit vorher durchzuführenden Reviews durch Kurator. Bereitstellung von zugehörigen Nutzerschnittstellen zur Unterstützung des Kurators und persistentem Reviewprozess mit der Möglichkeit der zeitweisen Unterbrechung.

3.1.2.2 Änderungsmanagement

Für das Management von Änderungen ist stets eine Nachvollziehbarkeit der durchgeführten Änderungen zu garantieren und die Möglichkeit des Rücksprungs auf einen vorherigen Stand bereitzustellen [Noy+06]. Wie bereits in Abschnitt 3.1.2.1 herausgestellt, sind zumeist mehrere Nutzer an der Entwicklung beteiligt. Dies macht wiederum die Nutzung von Entwicklungszweigen und die Erstellung von Releases notwendig, wie dies auch bei

etablierten Revisionsverwaltungssystemen umgesetzt wird. Die einzelnen Revisionen müssen dabei miteinander vergleichbar sein, sodass abgeleitet werden kann, was hinzugefügt, was gelöscht und was modifiziert wurde [Noy+06]. Diese gefundenen Unterschiede müssen wiederum entsprechend für den Nutzer aufbereitet werden, sodass dieser diese Information für die Zusammenführung von divergierten Entwicklungszweigen nutzen kann [Lev+10]. In diesem Zusammenhang entsteht auch die Forderung nach einer Möglichkeit, Konflikte detektieren und auflösen zu können, die bei der Zusammenführung auftreten können. Hierbei ist der Kontext der einzelnen zusammenzuführenden Revisionen essenziell. So können nicht nur einzelne Elemente auf Konflikte untersucht werden, sondern es müssen beispielsweise auch Vererbungsbeziehungen berücksichtigt werden [Noy+06]. Kehrer *et al.* [Keh+12] beschreiben in diesem Zusammenhang auch die Notwendigkeit des Wissens über den semantischen Effekt einer Evolution, um entsprechend auf diese reagieren zu können. Hierfür muss die Semantik der Evolution extrahiert werden. Das wird auch durch Ruhroth *et al.* [Ruh+14] bestätigt, die die Abstraktion von atomaren Änderungen fordern, was in einer Beschreibung durch High-Level-Changes resultiert. Eine solche Beschreibung wird beispielsweise auch durch Papavasileiou *et al.* [Pap+13] eingeführt. High-Level-Changes entsprechen demnach der Aggregation von atomaren Änderungen zu semantischen Änderungen. Diese semantischen Änderungen werden durch eine Gruppe von atomaren Änderungen identifiziert und stellen die Bedeutung der Gruppe der durchgeführten Änderung dar [Keh15]. High-Level-Changes repräsentieren daher in vielen Fällen Editieroperationen, die beispielsweise durch Nutzer durchgeführt werden [Sto04; Pap+13; Keh15; Hau+17; Pie+18].

Zur Erreichung der vorangegangenen Anforderungen ist eine semantische Beschreibung des Revisionsmodells unablässig. So müssen nach Noy *et al.* [Noy+06] Änderungskommentare, die den Beweggrund der Änderung angeben, beschrieben werden und Änderungen zwischen zwei Revisionen abfragbar sein, was die Kenntnis der Revisionshistorie voraussetzt. Des Weiteren müssen neue Revisionen erstellbar sein, wobei die zugehörige Vorgängerrevision angegeben werden muss. Überdies müssen neue Revisionen in Bezug auf den Vorgänger als abwärtskompatibel oder nicht abwärtskompatibel beschreibbar sein und stets erkennbar sein, wer welche Änderung mit welcher semantischen Auswirkung durchgeführt hat. Klein und Fensel [KF01] fordern weiterhin, dass Elemente eindeutig in der Revisionshistorie identifizierbar sein müssen und Relationen zwischen Revisionen explizit dargestellt werden müssen, um einen transparenten Zugriff zu ermöglichen.

Anforderungen inklusive Kurzbeschreibung

- Nachvollziehbarkeit von Änderungen

  Änderungen müssen stets nachvollziehbar gespeichert werden und jeweils zugreifbar sein. Mechanismen etablierter Revisionsverwaltungssysteme, wie Entwicklungszweige und Releases, müssen unterstützt werden und einzelne Revisionen miteinander vergleichbar sein.

- Zusammenführung divergierter Entwicklungszweige

  Divergierte Entwicklungszweige müssen zusammengeführt werden können. Dabei müssen Konflikterkennung und -lösung unterstützt werden.

- High-Level-Changes

  Atomare Änderungen müssen zu High-Level-Changes abstrahiert werden können, um die Semantik der Evolution darstellen zu können, sowie die Zusammenführung von divergierten Entwicklungszweigen zu unterstützen.

- Semantische Beschreibung des Revisionsmodells

  Das Revisionsmodell muss vollständig semantisch beschrieben werden. Zu Revisionen muss Metainformation anlegbar sein, wie zum Beispiel Kommentar, Ersteller, Vorgängerrevision und Abwärtskompatibilität.

3.1.2.3 Evolution

Klein und Frenzel [KF01] beschreiben neben der reinen Revisionsverwaltung des Weiteren die Notwendigkeit des Managements von durchgeführten Änderungen, wobei eine maximale Interoperabilität mit bestehenden Daten geschaffen werden soll. Hierfür können Co-Evolutionsstrategien genutzt werden, wie bereits in Abschnitt 2.3.1 eingeführt. Entsprechend müssen Modellrelationen zwischen Metamodellen und Modellen [DIP11], zwischen Typ und Instanz [DIP11] und zwischen Modell und Modell auf unterschiedlichen Abstraktionsniveaus [HHH14; Ruh+14] beachtet und Auswirkungen propagiert werden. Hierfür müssen die notwendigen Evolutionsschritte aus der Historie extrahiert und abstrahiert werden, was mittels der im Abschnitt 3.1.2.2 eingeführten High-Level-Changes möglich wird.

Die durchgeführten Evolutionsschritte müssen wiederum, ebenso wie jede andere Änderung, dokumentiert und nachvollziehbar gespeichert werden. Daraus ergibt sich, dass die Änderungen zwischen zwei Revisionen durch eine Reihe von Evolutionsschritten beschreibbar sein müssen [Ruh+14]. Das bietet die Grundlage, um die Evolution eines modellbasierten Systems zu verstehen [Keh+12].

Die möglichen Evolutionsschritte können beispielsweise über Regelsätze beschrieben werden. Hier besteht jedoch die Möglichkeit, dass diese anzuwendenden Regelsätze in Bezug auf die zugrunde liegende Revisionshistorie nicht vollständig sind. Daher wird durch Ruhroth [Ruh+14] gefordert, dass es eine Überprüfung gibt, ob durch die Anwendung der vorhandenen (Co-)Evolutionsschritte alle zugrunde liegenden Änderungen der Revisionshistorie abgedeckt werden.

Anforderungen inklusive Kurzbeschreibung

- Umsetzung von (Co-)Evolutionsstrategien

  (Co-)Evolutionsstrategien müssen so umgesetzt werden, dass Modellrelationen beachtet und zugehörige Auswirkungen propagiert werden.

- Semantische Beschreibung der (Co-)Evolutionsstrategien

  Angewendete (Co-)Evolutionsstrategien und deren zugrunde liegende Evolutionsschritte mit den zugehörigen Regelsätzen müssen semantisch beschrieben sein und in der Revisionshistorie referenziert werden.

- Vollständigkeitsprüfung der Evolutionsschritte

  Vor der Anwendung von (Co-)Evolutionsstrategien muss überprüft werden, ob alle zugrunde liegenden Änderungen der Revisionshistorie abgedeckt werden.

3.1.2.4 Semantische Modellbeschreibung

Für die im Abschnitt 3.1.2.2 beschriebenen High-Level-Changes muss das Informationsmodell beziehungsweise das Metamodell bekannt sein, um die Änderungen auf einer abstrahierten Ebene beschreiben zu können, wie beispielsweise in [Pap+13]. Diese können domänenübergreifend verwendet werden, wenn jeweils das gleiche Metamodell verwendet wird. Die zugrunde liegenden Regeln können aber auch so erweitert werden, dass domänenspezifische High-Level-Changes erkannt werden können, wodurch Änderungen für den Nutzer nachvollziehbarer werden. Als Grundlage hierfür kann die Semantik des Informationsmodells benutzt werden. Diese ist nach Lucas *et al.* [LMT09] außerdem wichtig, um bei der verteilten Entwicklung, die potenziell auch geografisch verteilt sein kann, unterschiedliche Interpretationen zu vermeiden.

Änderungen an einem Modell können Auswirkungen auf verbundene Modelle haben [RLL98; Lev+10]. Diese Auswirkungen müssen, wie in Abschnitt 3.1.2.3 aufgeführt, propagiert werden können. Die hierfür notwendigen Modellrelationen müssen wiederum semantisch beschrieben werden, damit diese bei der Evolution berücksichtigt werden können.

Anforderungen inklusive Kurzbeschreibung

- Semantische Beschreibung der revisionierten Modelle

  Die revisionierten Modelle müssen semantisch beschrieben sein. Hierfür können Informations- und Metamodelle genutzt werden.

- Semantische Beschreibung von Modellrelationen

  Verbindungen zwischen und innerhalb von Modellen müssen explizit dargestellt und semantisch beschrieben sein.

3.1.2.5 Qualitätsattribute

Für eine Umsetzung der bisherigen Anforderungen müssen weitere Qualitätsattribute beachtet werden. Ruhroth *et al.* [Ruh+14] sagen hierzu aus, dass die notwendigen Berechnungen transparent und automatisch zu erfolgen haben. Das heißt, dass Nutzer

mögliche Reinterpretationen von eigenen Aktionen nicht bemerken sollen und bei Entwicklungsarbeiten nicht unterbrochen werden, um durchgeführte Änderungen analysieren zu müssen.

Bei der Anwendung von (Co-)Evolutionsstrategien müssen wiederum die daraus resultierenden Änderungen verifiziert werden. Hierfür ist sicherzustellen, dass weitere Qualitätsattribute integrierbar und prüfbar sind. Levendovszky *et al.* [Lev+10] führen hierzu beispielsweise Prüfungen von Konsistenz und Vollständigkeit der Modelle auf, wobei diese auch durch weiterführende Qualitätsattribute wie Lesbarkeit und Evolvability ergänzt werden können. Noy und Klein [NK04] fügen hinzu, dass auch die Kompatibilität zwischen Revisionen prüfbar und darstellbar sein muss, um entsprechende Abwärtskompatibilität sicherstellen zu können.

Anforderungen inklusive Kurzbeschreibung

- Transparente und automatische Berechnung

  Die Umsetzung der Anforderungen soll transparent erfolgen, wobei die Nutzer keine Reinterpretationen ihrer Aktionen bemerken sollen und nicht für die Analyse von durchgeführten Änderungen unterbrochen werden.

- Unterstützung der Integration zusätzlicher Qualitätsattribute

  Zusätzliche Qualitätsattribute von Modellen sollen integriert werden können, um diese nach der Durchführung von Änderungen prüfen und auswerten zu können. Beispiele sind Konsistenz, Kompatibilität und Vollständigkeit.

3.1.3 Anwendungsfälle

Für diese Arbeit wird die Co-Simulation und die Modularisierung, hierbei insbesondere die Spezifikation des MTP, als Anwendungsfall genutzt. Eine allgemeine Einführung in diese beiden Themenbereiche erfolgt in Abschnitt 1.4. Im Folgenden werden die beiden Anwendungsfälle detaillierter auf ihre jeweiligen Problemstellungen in Bezug auf die Evolution von Informationsmodellen analysiert.

3.1.3.1 Co-Simulation

Bei der kollaborativen Erstellung von Co-Simulationsensembles werden unterschiedliche Simulationsmodelle zu einer Gesamtsimulation zusammengeschaltet. Ein wesentlicher Aspekt bei der Erstellung und der anschließenden Pflege ist der Umgang mit Änderungen an den Modellen und den Verschaltungen zwischen diesen. Auf Basis der Verschaltungen können während der Laufzeit der Gesamtsimulation die Kommunikationswege zwischen den Einzelsimulationen abgeleitet werden.

Modellierung eines Co-Simulationsensembles Ein Beispiel für einen Co-Simulationsstandard ist das Functional Mock-up Interface (FMI) [Mod10]. Dieser Standard spezifiziert jedoch nur die Schnittstelle zu den Simulationseinheiten, die als Functional

Mock-up Unit (FMU) bezeichnet werden. Jede Einzelsimulation stellt entsprechend Ports zur Verfügung, die dann miteinander verschaltet werden können. Der Nutzer ist selbst sowohl dafür verantwortlich wie die Verbindungen zwischen den Einzelsimulationen definiert und aufgebaut werden, als auch dafür, wie die Steuerung des Ensembles realisiert wird [Gal+15]. Zur Erstellung eines geeigneten Masteralgorithmus, der die Steuerung übernimmt, ist es notwendig, die Abhängigkeiten zwischen den Einzelsimulationen zu analysieren [Cam+16]. Hierfür müssen diese entsprechend auswertbar vorliegen. Bastian *et al.* fordern dafür einen gerichteten Graphen, in dem Knoten Simulatoren und Kanten auszutauschende Daten darstellen [Bas+11]. Des Weiteren sollten auch interne Abhängigkeiten zwischen Eingangs- und Ausgangsports modellierbar sein [Van+15]. FMI definiert bereits die Attribute, die zur Beschreibung von Einzelsimulationen und deren Ports notwendig sind. Datentypen und Einheiten werden ebenso durch FMI beschrieben. Hier können aber auch andere Standards zum Einsatz kommen, wie beispielsweise die semantische Beschreibung von Quantities, Units, Dimensions, and Data Types (QUDT)[1)]. Vor allem bei kontinuierlichen Prozesssimulationen beinhalten die Verbindungen zwischen den Modellen nicht nur einen Wert, der ausgetauscht werden muss. So müssen zum Beispiel zur Modellierung einer Rohrleitung Durchfluss und Energie zwischen den Simulationen übertragen werden. Hierfür kann das Konzept der Bondgraphen [Bre11] wiederverwendet werden. Bestehende Tools verwenden zumeist ein internes proprietäres Modell zur Beschreibung des Ensembles [Cam+16]. Dieses ist nur durch das spezifische Tool auswertbar, wodurch alternative Masteralgorithmen es nicht auswerten können.

Möglichkeiten der Verschaltung Die Verschaltung von Einzelsimulationen wird zumeist über einfache Konfigurationsdateien gelöst. Beispielsweise wird eine Textdatei zur Konfiguration des Masters genutzt, in der auch die vorhandenen Simulationen und die korrespondierenden Verschaltungen enthalten sind [Bas+11]. Das setzt wiederum voraus, dass alle Simulatoren bereits verfügbar sind, alle Verschaltungen zu einem definierten Zeitpunkt fest definiert sein müssen und auf Basis dieser Konfiguration alle Verschaltungen erzeugt werden. Abbildung 3.1 stellt das prinzipielle Vorgehen am Beispiel von vier Simulatoren vor, die in Schritt 1 alle verfügbar sind und in Schritt 2 miteinander verschaltet werden.

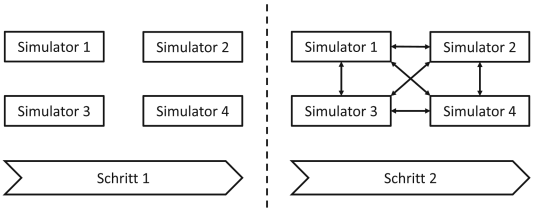


Abbildung 3.1: Verschaltung eines Co-Simulationsensembles mit bekannten Einzelsimulationen

[1)] http://www.qudt.org/ (besucht am 29.11.2020)

Im Gegensatz zu einer Gesamtkonfiguration des Ensembles besteht auch die Möglichkeit, dass in einer kollaborativen Umgebung mehrere Nutzer jeweils ihren eigenen Simulator mit anderen bereits verfügbaren Simulatoren verschalten möchten. Die Abbildung 3.2 gibt hierzu wiederum ein strukturiertes Beispiel, in dem die einzelnen Simulatoren nacheinander verfügbar sind und jeweils nur Verschaltungen zu anderen bereits verfügbaren Simulatoren definieren. In einer solchen kollaborativen Umgebung sind jedoch auch weitere Möglichkeiten denkbar, bei denen bestehende Simulatoren neu hinzugekommene verschalten oder Verschaltungen iterativ hinzugefügt und bearbeitet werden. In einem solchen Szenario muss eine Nachvollziehbarkeit der durchgeführten Änderungen gegeben sein und auch eine persistente Speicherung gewährleistet werden, um eine Konfiguration später auch wiederherstellen zu können.

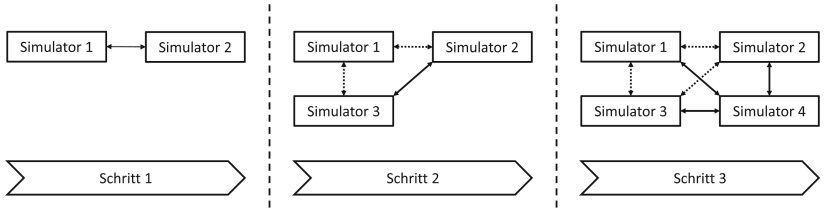


Abbildung 3.2: Kollaboratives Verschalten innerhalb eines Co-Simulationsensembles

Änderungen innerhalb des Co-Simulationsensembles Im vorherigen Abschnitt wurde auf zwei Möglichkeiten der Verschaltung von Simulationen eingegangen. Insbesondere in kollaborativen Umgebungen ist ein wesentlicher Aspekt bei der Erstellung und der Pflege eines Co-Simulationsensembles der Umgang mit Änderungen an Einzelsimulationen und den Verschaltungen zwischen diesen. Nachfolgend werden beispielhaft drei wesentliche Möglichkeiten für Änderungen dargestellt. Es wird dabei jeweils auf die Auswirkungen der durchgeführten Änderungen auf die Liste der durch die Einzelsimulation angebotenen Ports (Portliste), die Verschaltungen zwischen den Ports (Portverschaltungen) und die gekoppelte Simulation eingegangen. Bei den nachfolgenden Betrachtungen wir immer von einem zumindest teilweise verschalteten Co-Simulationsensemble ausgegangen, da Änderungen an nicht verschalteten Ports keine direkten Auswirkungen auf andere Simulationen haben.

Abbildung 3.3 zeigt eine simple Änderung des Namens eines Ports. Port E von Simulator B wird zu Z umbenannt. Diese Änderung wird innerhalb von Simulator B vorgenommen und sollte automatisch von der Portliste von Simulator B übernommen werden. Je nach Umsetzung der Co-Simulationsumgebung muss nachfolgend entweder die Portverschaltung angepasst werden, da die Ports direkt über deren Namen identifiziert werden, oder wenn eine zusätzliche eindeutige Identifizierung der Ports genutzt wird, so muss keine Änderung an der Verschaltung vorgenommen werden. In jedem Fall hat die Änderung keine Auswirkung auf den gekoppelten Simulator.

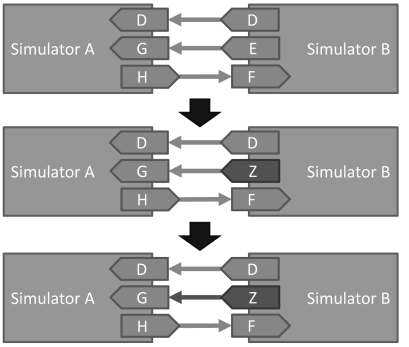


Abbildung 3.3: Beispiel für die Änderung einer Portbezeichnung

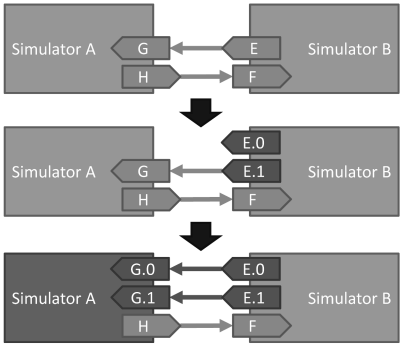


Abbildung 3.4: Beispiel für die Auftrennung eines Ports in mehrere Ports

Eine weitere Möglichkeit ist die Auftrennung eines Ports in mehrere Ports. Zum Beispiel könnte ein Port, der ein Statuswort bereitstellt, in mehrere einzelne Ports aufgespalten werden, wobei jeder Einzelport einen booleschen Wert beschreibt. In Abbildung 3.4 ist ein Beispiel dargestellt, bei dem Port E von Simulator B in zwei Ports E.0 und E.1 aufgespalten wird. Diese Änderung sollte wiederum automatisch von der Portliste von Simulator B übernommen werden, indem der alte Port durch zwei neue Ports ersetzt wird. In diesem Fall muss auch die Verschaltung angepasst werden. In dem dargestellten Beispiel wird sie durch zwei neue Verschaltungen ersetzt, die die Verschaltung zu neu erstellten Ports in Simulator A beschreiben. Die Ports G.0 und G.1 werden aufgrund der durchgeführten Änderungen in der Portliste erstellt und müssen dementsprechend auch in die Simulation eingepflegt werden. Ebenso besteht die Möglichkeit des umgekehrten Falls, bei dem mehrere Ports zu einem Port zusammengefasst werden oder das andere bereits existente Ports für die neue Verschaltung genutzt werden können, ohne das zusätzliche Ports erstellt werden müssen.

In den Abbildungen 3.5 und 3.6 ist die Änderung des Datentyps eines Ports anhand von zwei möglichen Beispielen dargestellt. Ausgangspunkt ist jeweils die Änderung des Datentyps von Port E von Simulator B. Im ersten Fall, dargestellt in 3.5, wird diese Änderung an den verschalteten Port G von Simulator A propagiert. Es ist daher keine Änderung an der Verschaltung notwendig, jedoch muss die Änderung in das Simulationsmodell von Simulator A eingepflegt werden und danach die Portliste von Simulator A mit dem neuen Datentyp von Port G aktualisiert werden. Abbildung 3.6 zeigt den zweiten Fall, in dem keine Änderung am Simulationsmodell von Simulator A notwendig ist, da aufgrund der Änderung eine Änderung an der Verschaltung vorgenommen wird. Sie wird auf einen anderen Port von Simulator A bezogen, der den gleichen Wert in einem anderen Datentyp darstellt.

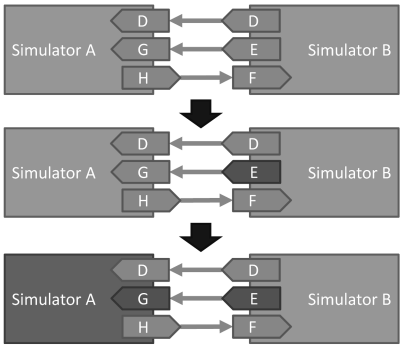


Abbildung 3.5: Beispiel für die Änderung des Datentyps eines Ports mit resultierender Änderung im verschalteten Simulator (veröffentlicht in [HGU18, S. 46])

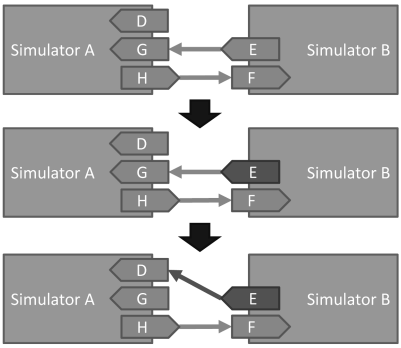


Abbildung 3.6: Beispiel für die Änderung des Datentyps eines Ports mit Änderung des Zielports des verschalteten Simulators (veröffentlicht in [HGU18, S. 46])

Schlussfolgerung In den vorangegangenen Abschnitten sind die wesentlichen Herausforderungen bei der Konfiguration und der Pflege einer Co-Simulationsumgebung anhand von Beispielen dargestellt. Es wird deutlich, dass ohne Unterstützung von zusätzlichen Tools hohe manuelle Aufwände notwendig sind. So entstehen bei der Kopplung von unterschiedlichen Applikationen und Softwaresystemen hohe Zeitaufwände [KVN12]. Erste Ansätze, wie [KVN12] und [Hen+16a; Hen+16b] nutzen Möglichkeiten der semantischen Beschreibung, um die Integration von unterschiedlichen Modellen zu vereinfachen. Weiterhin werden von Karhela *et al.* [KVN12] Revisionsverwaltungsmechanismen gefordert, um Konfigurationen und Simulationsergebnisse nachvollziehbar zu speichern. Der Umgang mit Änderungen und die Vermeidung von manuellen Aufwänden in diesem Zusammenhang wird jedoch nicht betrachtet.

3.1.3.2 Modularisierung

Die Standardisierung des MTP erfolgt in der VDI/VDE/NAMUR-Richtlinienserie 2658, die so aufgebaut ist, dass nach und nach neue Aspekte in neuen Blättern der Serie spezifiziert werden. Dies ist dem agilen Ansatz der Spezifikation zuträglich, da nach jeder Iteration die Ergebnisse bereits in die Standardisierung überführt werden können. In einem solchen Ansatz muss jedoch auch mit Änderungen von bestehenden Blättern und der Ergänzung von zusätzlichen Blättern umgegangen werden, um stets ein konsistentes Gesamtgefüge der Richtlinien bereitstellen zu können. Gleiches gilt für die informationstechnische Beschreibung der Aspekte in den Richtlinien.

Engineering Workflow Das Engineering von modularen Anlagen unterteilt sich nach [VDI17] in zwei Bereiche, wie in Abbildung 3.7 dargestellt. Zum einen in ein projektunab-

hängiges Modulengineering, in dem Module entwickelt und hergestellt werden und zum anderen in ein projektbezogenes Anlagenengineering, bei dem Module zu einer Gesamtanlage integriert werden. Der Modulhersteller liefert zu jedem Modul ein MTP mit, dass aus den Engineeringdaten heraus erzeugt wird. Dieses MTP wird mit dem physikalischen Modul ausgeliefert. Das Modul wird dann physikalisch und informationstechnisch in die Gesamtanlage integriert. Für die informationstechnische Integration steht das MTP zur Verfügung, das alle erforderlichen Aspekte für die Integration in eine übergeordnete Prozessführungsebene (PFE)[2)] beschreibt, in der dann die Orchestrierung der modularen Anlage erfolgen kann. Die Orchestrierung kann beispielsweise wie in [Blo+17] beschrieben realisiert werden.

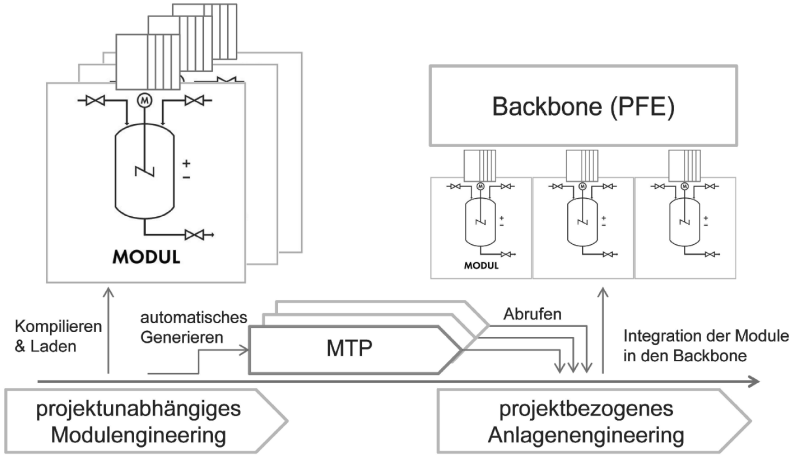


Abbildung 3.7: Engineering Workflow modularer Anlagen [VDI17, S. 6]

Aufbau des MTP Das MTP ist derzeit als eine Containerdatei realisiert, die die einzelnen zu beschreibenden Aspekte zusammenfasst. Die Definition der notwendigen Informationsmodelle erfolgt hierbei technologieunabhängig durch UML-Modelle, wobei jeweils auch eine konkrete Umsetzung auf dem Extensible Markup Language (XML) basierten Automation Markup Language (AutomationML) Standard (in Anlehnung an [Hoe+16]) beschrieben wird. Blatt 1 [VDI17] der Serie definiert hierfür den allgemeinen Aufbau und die notwendige Kommunikationsstruktur. Im ersten Schritt wird für die Kommunikation jeweils OPC UA verwendet. Zentrales Element der Beschreibung ist das sogenannte Manifest, das Verweise auf die einzelnen Aspekte im MTP enthält. Zur Beschreibung der weiteren Aspekte werden jeweils UML-Diagramme, erklärender Text

[2)] Die Bezeichnung PFE wird in dieser Arbeit synonym zu Process Orchestration Layer (POL) (definiert in [VDI17]) verwendet.

und Umsetzungsbeispiele in AutomationML verwendet. Ein Beispiel für einen weiteren Aspekt ist die Bedienbildbeschreibung, die in Blatt 2 [VDI18] spezifiziert ist.

Mögliche zukünftige Entwicklungen Das MTP beschreibt einen Modultypen und kann daher für unterschiedliche konkrete Instanzen eingesetzt werden, wobei der Anlagenbetreiber das MTP vom Modulhersteller noch vor der Lieferung des Moduls fordern kann, um es bereits in die PFE informationstechnisch zu integrieren. Es ist jedoch auch denkbar, dass zukünftig der Anlagenbetreiber dem Modulhersteller ein MTP als Spezifikation für ein herzustellendes Spezialmoduls übergibt. Im Fall, dass ein bereits in einer Anlage integriertes Modul durch ein anderes ausgetauscht wird, was die gleiche Funktionalität aber beispielsweise einen anderen Aufbau des OPC UA Servers besitzt, muss in der zugehörigen Modulverwaltungssoftware eine weitere Abstraktion geschaffen werden, um eine Abbildung zwischen unterschiedlichen MTPs zu ermöglichen.

Der durch das Modul bereitgestellte OPC UA Server kann zukünftig auch direkt das Modell des Moduls bereitstellen, ohne das eine Offlinedatei, wie das MTP, ausgetauscht werden muss. Die notwendige Information ist in diesem Fall dann direkt im OPC UA Informationshaushalt abgebildet und kann durch eine PFE ausgelesen werden, sobald das Modul physikalisch in die Anlage integriert wird. Eine zusätzliche Typbeschreibung kann dann beispielsweise direkt aus der Typenbeschreibung in OPC UA abgeleitet werden. Wird die Beschreibung bereits früher benötigt, so kann zum Beispiel ein ausführbarer OPC UA Server oder ein entsprechendes Abbild geliefert werden.

Eine Virtuelle Inbetriebnahme (VIBN) benötigt ebenso Simulationsmodelle, um durchgeführt werden zu können. Diese können wiederum als ein eigener Aspekt im MTP beschrieben werden. Die Simulationsmodelle können dann beispielsweise direkt auf den Modulen verfügbar sein oder die Simulationen werden über Clouddienste zur Verfügung gestellt. In beiden Fällen müssen wiederum Möglichkeiten geschaffen werden, um die Simulationen zu koppeln und auszuführen.

Schlussfolgerung Die Verfolgung eines solch agilen Ansatzes, wie er in der Standardisierung des MTP Anwendung findet, hat Geschwindigkeitsvorteile und die Ergebnisse sind belastbarer, da bereits Prototypen bestehen. Es muss jedoch mit Änderungen umgegangen werden, die während der Iterationen und auch nachfolgend vorgenommen werden. Ziel muss es daher sein, stets ein konsistentes Gesamtgefüge der einzelnen Richtlinienblätter der Serie zu erreichen. Hierdurch entstehen mehrere Versionen eines MTP, die jeweils einen Entwicklungsstand kennzeichnen und in denen unterschiedliche Aspekte bereits verfügbar sind. Ebenso kann es vorkommen, dass nicht alle Hersteller mit der gleichen Geschwindigkeit auf die Änderungen und Erweiterungen in den Richtlinien reagieren können, wodurch es vorkommen kann, dass unterschiedliche Versionen des MTP am Markt angeboten werden. Hierfür werden Migrationsstrategien benötigt, um beispielsweise ein bestehendes MTP einer bestimmten Version auf eine aktuelle zu migrieren.

3.1.4 Anforderungen

Nachfolgend werden die in den vorangegangenen Abschnitten aufgenommenen Anforderungen tabellarisch in Tabelle 3.1 dargestellt. Jeder Anforderung wird dabei ein eindeutiger Identifikator zugeordnet. Dieser dient im weiteren Verlauf der Abkürzung der Beschreibung und der Identifizierung innerhalb des Dokuments. Die Analyse der beiden Anwendungsfälle zeigt, dass die allgemein aufgenommenen Anforderungen die Problemstellungen der Anwendungsfälle bereits abdecken, wie in Tabelle 3.2 dargestellt. Daher ist keine Erweiterung der Anforderungen notwendig.

In Tabelle 3.2 ist des Weiteren dargestellt, durch welchen Anwendungsfall welche Anforderungen abgedeckt sind. Anforderungen, die direkt aus einem Anwendungsfall abgeleitet werden können, sind mit ✓ markiert und Anforderungen, die nicht auf den Anwendungsfall zutreffen, sind mittels ✗ beschrieben. In Klammern ist die Markierung angegeben, wenn es sich bei der Anforderung um eine allgemeine Anforderung handelt, die jedoch nicht im ersten Schritt aus dem jeweiligen Anwendungsfall abgeleitet beziehungsweise ausgeschlossen werden kann.

Zusammenfassend kann festgehalten werden, dass Anforderungen des Nutzungskontextes vor allem in Bezug auf die Co-Simulation zutreffen, da es sich dabei überwiegend um kollaborative Szenarien handelt, wohingegen sich die Migration von MTP-Versionen im ersten Schritt auf die Migrationsmechanismen und weniger auf den Nutzungskontext bezieht. Bei A-101 handelt es sich um eine generelle Anforderung, die beiden Anwendungsfällen zugeordnet werden kann. Die restlichen Anforderungen treffen jeweils auf beide Anwendungsfälle zu, wobei der Fokus bei der Durchführung von Evolutionen bei der Co-Simulation vorrangig auf Verbindungen auf Instanzebene liegt und bei der Modularisierung auf Typenmodelländerungen. Es ist jedoch nicht auszuschließen, dass jeweils weitere Änderungen auf Instanz- beziehungsweise Typenebene auftreten.

Tabelle 3.1: Anforderungszusammenfassung

ID	Anforderung	Beschreibung
A-100	Nutzungskontext	
A-101	Selbstbeschreibungsfähigkeit des Systems	Die Nutzung des Systems soll unabhängig von Vorkenntnissen von unterschiedlichen Nutzern genutzt werden können.
A-102	Rollenmanagement für Nutzer	Das System unterstützt die Verwaltung und Nutzung von unterschiedlichen Rollen, die am Evolutionsprozess beteiligt sind.
A-103	Zugriffsmanagement mit Rechtevergabe	Beschränkung von Zugriffsrechten und Verwaltung von Rechten für Nutzer, Nutzergruppen und Nutzerrollen.

Tabelle 3.1: Anforderungszusammenfassung *(Fortsetzung)*

ID	Anforderung	Beschreibung
A-104	Umsetzung von Freigabeprozessen	Möglichkeit der Etablierung von Freigabeprozessen mit vorher durchzuführenden Reviews durch Kurator. Bereitstellung von zugehörigen Nutzerschnittstellen zur Unterstützung des Kurators und persistentem Reviewprozess mit der Möglichkeit der zeitweisen Unterbrechung.
A-200	Änderungsmanagement	
A-201	Nachvollziehbarkeit von Änderungen	Änderungen müssen stets nachvollziehbar gespeichert werden und jeweils zugreifbar sein. Mechanismen etablierter Revisionsverwaltungssysteme, wie Entwicklungszweige und Releases, müssen unterstützt werden und einzelne Revisionen miteinander vergleichbar sein.
A-202	Zusammenführung divergierter Entwicklungszweige	Divergierte Entwicklungszweige müssen zusammengeführt werden können. Dabei müssen Konflikterkennung und -lösung unterstützt werden.
A-203	High-Level-Changes	Atomare Änderungen müssen zu High-Level-Changes abstrahiert werden können, um die Semantik der Evolution darstellen zu können sowie die Zusammenführung von divergierten Entwicklungszweigen zu unterstützen.
A-204	Semantische Beschreibung des Revisionsmodells	Das Revisionsmodell muss vollständig semantisch beschrieben werden. Zu Revisionen muss Metainformation anlegbar sein, wie zum Beispiel Kommentar, Ersteller, Vorgängerrevision und Abwärtskompatibilität.
A-300	Evolution	
A-301	Umsetzung von (Co-)Evolutionsstrategien	Die (Co-)Evolutionsstrategien müssen so umgesetzt werden, dass Modellrelationen beachtet und zugehörige Auswirkungen propagiert werden.

Tabelle 3.1: Anforderungszusammenfassung *(Fortsetzung)*

ID	Anforderung	Beschreibung
A-302	Semantische Beschreibung der (Co-)Evolutionsstrategien	Angewendete (Co-)Evolutionsstrategien und deren zugrunde liegende Evolutionsschritte mit den zugehörigen Regelsätzen müssen semantisch beschrieben sein und in der Revisionshistorie referenziert werden.
A-303	Vollständigkeitsprüfung der Evolutionsschritte	Vor der Anwendung von (Co-)Evolutionsstrategien muss überprüft werden, ob alle zugrunde liegenden Änderungen der Revisionshistorie abgedeckt werden.
A-400	Semantische Modellbeschreibung	
A-401	Semantische Beschreibung der revisionierten Modelle	Die revisionierten Modelle müssen semantisch beschrieben sein. Hierfür können Informations- und Metamodelle genutzt werden.
A-402	Semantische Beschreibung von Modellrelationen	Verbindungen zwischen und innerhalb von Modellen müssen explizit dargestellt und semantisch beschrieben sein.
A-500	Qualitätsattribute	
A-501	Transparente und automatische Berechnung	Die Umsetzung der Anforderungen soll transparent erfolgen, wobei die Nutzer keine Reinterpretationen ihrer Aktionen bemerken sollen und nicht für die Analyse von durchgeführten Änderungen unterbrochen werden.
A-502	Unterstützung der Integration zusätzlicher Qualitätsattribute	Zusätzliche Qualitätsattribute von Modellen sollen integriert werden können, um diese nach der Durchführung von Änderungen prüfen und auswerten zu können. Beispiele sind Konsistenz, Kompatibilität und Vollständigkeit.

Tabelle 3.2: Abgedeckte Anforderungen pro Anwendungsfall

	Anforderungen		Co-Simulation	Modularisierung
Nutzungskontext	A-101	Selbstbeschreibungsf.	(✓)	(✓)
	A-102	Rollenmanagement	✓	(✗)
	A-103	Zugriffsmanagement	✓	(✗)
	A-104	Freigabeprozesse	✓	(✗)
Änderungsmanagement	A-201	Nachvollziehbarkeit	✓	✓
	A-202	Zusammenführung	✓	✓
	A-203	High-Level-Changes	✓	✓
	A-204	Revisionsmodell	(✓)	(✓)
Evolution	A-301	Strategien	✓	✓
	A-302	Sem. Beschr.	(✓)	(✓)
	A-303	Vollständigkeitsp.	✓	✓
Sem. Modellbeschr.	A-401	Rev. Modelle	(✓)	(✓)
	A-402	Verbindungen	✓	✓
Qualitätsattribute	A-501	Berechnung	(✓)	(✓)
	A-502	Integration	✓	✓

3.2 Analyse bestehender Ansätze

Im Folgenden werden bestehende Ansätze gegen die aufgenommenen Anforderungen abgeglichen. Es werden dabei erste Ansätze für das Management von Evolution [Keh15; Sto04; Ruh+14; Bür+14], für die Bereitstellung einer Integrationsplattform [KVN12] und für die Revisionsverwaltung [Noy+06; GHU14] vorgestellt. Hierzu erfolgt zuerst eine Kurzdarstellung der jeweiligen Ansätze. Anschließend werden die Ansätze in Tabelle 3.3 in Hinblick auf die Umsetzung der Anforderungen bewertet.

3.2.1 Dissertation Timo Kehrer [Keh15]

Kehrer [Keh15] stellt einen Ansatz zur Hebung von Modelldifferenzen auf ein Nutzerlevel vor, um Modellierer beim Erkennen und Verwalten von Änderungen besser unterstützen zu können. Kernelement sind sogenannte Editieroperationen, die in vielen Standardmodelleditoren oder Refactoringtools Verwendung finden. Grundlage für die Erkennung dieser Editieroperationen ist eine formale Spezifikation der Operationen unter Nutzung der Graphtransformationssprache Henshin[3)], die speziell für das Eclipse Modeling Framework (EMF) entwickelt wurde. Des Weiteren werden die Low-Level-Changes benötigt, anhand derer die Operationen erkannt werden. Diese Low-Level-Changes werden durch Modellvergleich erzeugt. Der Ansatz wurde innerhalb von SiLift[4)] umgesetzt.

Der Ansatz bietet eine gute Übersicht zu den Grundlagen für die Erstellung von Regelsätzen für die Erkennung von High-Level-Changes. Der Einbezug von Modellhistorien für die Erkennung wird nur im Ausblick aufgeführt. Dies liegt auch daran, dass in dem Konzept kein semantisches Revisionsverwaltungssystem vorgesehen ist. Es wird wiederum im Ausblick darauf verwiesen, dass ein etabliertes System als Grundlage dienen soll. Hierdurch würde jedoch die Semantik der Modelländerungen verloren gehen. Co-Evolutionen werden ebenso nur in beschränktem Umfang betrachtet. Es werden nur Propagierungen von Modelländerungen an andere Varianten eines Modells beachtet, was einem einfachen Patchverfahren entspricht, bei dem erkannte Editieroperationen an einem Modell ebenso auf ein anderes angewendet werden.

3.2.2 Dissertation Ljiljana Stojanovic [Sto04]

In [Sto04] werden von Stojanovic Methoden und Tools für die Evolution von Ontologien entwickelt. Insbesondere wird ein Prozess beschrieben, der eine effiziente Ontologieevolution erlaubt. Dieser besteht aus der Handhabung von Ontologieänderungen, der Absicherung der Konsistenz von Ontologien unter Beachtung von Abhängigkeiten und unterstützt die Nutzer bei der Verwaltung von Änderungen. Des Weiteren werden dem Nutzer Hinweise für ein kontinuierliches Ontologiereengineering gegeben. Besonderer Wert wird auf die Anwendbarkeit der Ansätze im Semantic Web gelegt. Hierbei findet vor allem die hohe Anzahl von Ontologien und deren physikalische Verteilung Beachtung.

[3)] https://www.eclipse.org/henshin/ (besucht am 29.11.2020)
[4)] http://pi.informatik.uni-siegen.de/Projekte/SiLift/ (besucht am 29.11.2020)

Die Implementierung erfolgt innerhalb vom Karlsruhe Ontology and Semantic Web framework (KAON)[5)].

Diese Arbeit stellt neben sogenannten Elementaränderungen und daraus zusammengesetzten Änderungen auch eine Evolutionsontologie dar, die die durchgeführten Änderungen semantisch für jede Ontologie beschreibt. Hierbei handelt es sich um Logs, die gespeichert werden. Diese sind ähnlich zu einem Revisionsverwaltungssystem, wobei grundlegende Funktionalitäten, wie die Zusammenführung von divergierten Zweigen oder die Erstellung von Versionen, nicht betrachtet werden. Evolutionen sind nur auf Ontologieebene umgesetzt, wenn es sich um Relationen zwischen Ontologien handelt. Das Instanzniveau wird daher nicht weiter betrachtet. Die beschriebenen High-Level-Änderungen sind insbesondere für das KAON umgesetzt, jedoch ist eine Abbildung der Ansätze auf beispielsweise die OWL nicht realisiert. Die Implementierung kann aber mit hohen Datenmengen umgehen. In Bezug auf die Anforderungen im Bereich des Nutzungskontext werden keine Nutzerrollen, keine Zugriffsrechte und keine Freigabeprozesse betrachtet.

3.2.3 SecVolution

SecVolution [Ruh+14; Bür+14] ist ein modellbasierter Ansatz aus dem Bereich der Sicherheit von Informationssystemen. Ziel ist es, die Sicherheit eines Informationssystems stets zu gewährleisten, auch dann, wenn Änderungen in der Umgebung des Systems die Sicherheit gefährden. Auf Basis der Änderungen und der Nutzung von internen und externen Wissensquellen erfolgt eine Anpassung der entsprechenden Softwaremodelle des Informationssystems, um das Sicherheitslevel des Systems wiederherzustellen. Hierfür werden Mechanismen der Co-Evolution angewendet.

Der grundlegende Informationsfluss im SecVolution-Ansatz ist in Abbildung 3.8 dargestellt. Änderungen in der Umgebung werden verfolgt und daraus notwendige Anpassungen an den Systemmodellen berechnet, die mittels Co-Evolutionen umgesetzt werden. Basis bildet hierfür das Security Maintenance Model, da dieses zum einen beschreibt, wie sich das sicherheitsrelevante Wissen weiterentwickeln kann und wie zum anderen die Co-Evolution der auf diesem Wissen aufbauenden Modelle durchzuführen ist. Da die Änderungen als gegeben angenommen werden, ist in diesem Ansatz kein Revisionsverwaltungssystem integriert. Die durchzuführenden Co-Evolutionen beziehen sich auf Typ-Instanz-Beziehungen, wobei Abhängigkeiten zwischen unterschiedlichen Modellen keine Beachtung finden. Die durchgeführten Evolutionen sollen jedoch semantisch beschrieben werden. In Bezug auf die Anforderungen im Bereich des Nutzungskontext werden Nutzerrollen aufgeführt, wie in Abbildung 3.8 dargestellt, jedoch werden Zugriffsrechte und notwendige Freigabeprozesse nicht explizit aufgeführt.

[5)] http://kaon2.semanticweb.org (besucht am 29.11.2020)

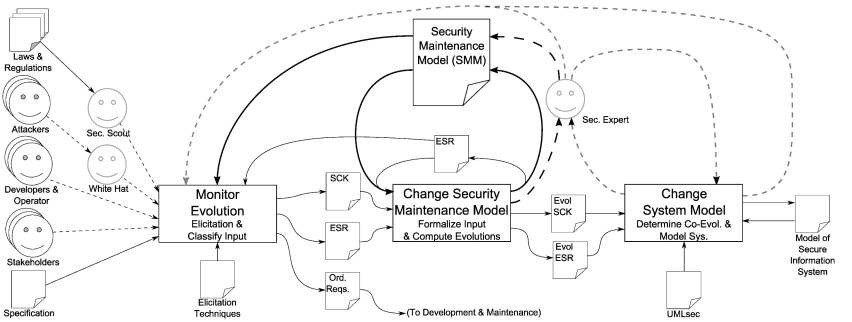


Abbildung 3.8: Informationsflussdiagramm des SecVolution-Ansatzes [Bür+14, S. 3]

3.2.4 Simantics

Das in [KVN12] vorgestellte Simantics[6)] ist eine offene Plattform die die Integration von unterschiedlichen Tools ermöglicht. Ziel ist es, eine integrierte Umgebung für die Modellierung und Simulation auf Basis einer vollständig semantischen Modellierung aufzubauen. Die Integration der spezifischen Applikationsmodelle erfolgt unter Verwendung von ontologiebasierten Abbildungen und Transformationen, die jeweils innerhalb des Frameworks definiert werden. Zum Einsatz kommt ein eigens entwickelter Triple Store, der speziell auf die hohen Anforderungen an das Management von großen Datensätzen und eine unterliegende Revisionskontrolle zugeschnitten ist.

Da es sich bei diesem Ansatz vorrangig um eine Integrationsplattform handelt, werden keine Mechanismen zur Evolution der integrierten Modelle beschrieben. Die Nachvollziehbarkeit ist durch die in den Triple Store eingebauten Revisionsverwaltungsfunktionalitäten gegeben, wobei jedoch keine High-Level-Changes beschrieben werden. Die Modellierung basiert auf einer eigens entwickelten Beschreibungssprache für Ontologien, die *Layer0* genannt wird und zu Teilen Gemeinsamkeiten mit OWL hat. Einzelne Modelle können des Weiteren miteinander verlinkt werden. Besonderes Augenmerk wird auf die Definition der Rollen gelegt, die die Plattform nutzen. Derzeit existieren jedoch keine Prozesse zur Freigabe von Ontologien oder für die Einschränkung von Zugriffsrechten der jeweiligen Rollen.

3.2.5 Changes Tab

Noy *et al.* [Noy+06] haben ein System für die Unterstützung der Ontologieevolution entwickelt, dessen zugrunde liegende Architektur in Abbildung 3.9 dargestellt ist. Zentrales Element ist die Change and Annotation Ontology (CHAO), mittels derer alle Änderungen an einer Ontologie beschrieben werden können. Änderungen können entweder direkt während der Bearbeitung oder durch den Vergleich von zwei Versionen aufgenommen

[6)] https://www.simantics.org/ (besucht am 29.11.2020)

werden. Das Framework ist als Plug-in mit dem Namen *Changes Tab*[7] für die Ontologieentwicklungsumgebung Protégé[8] realisiert und besteht aus zwei Teilen. Zum einen aus einem Plug-in, um Änderungen verwalten zu können und mithilfe dessen die Listen der Änderungen eingesehen, Annotationen vorgenommen und Änderungen innerhalb der CHAO aufgenommen werden können. Zum anderen wird mittels des PROMPT Plug-ins der Vergleich von zwei Versionen und die Akzeptierung und Ablehnung von einzelnen Änderungen ermöglicht.

Bei diesem System handelt es sich um eine reine Änderungsverfolgung, die keine weitreichenderen Revisionsverwaltungsfunktionalitäten, wie beispielsweise Entwicklungszweige, unterstützt. Die aufgenommenen Änderungen werden jedoch vollständig semantisch durch die CHAO beschrieben und können des Weiteren gruppiert werden. Hierdurch entstehen, neben den bereits aus den Editieroperationen in Protégé abgeleiteten, weitere High-Level-Changes, die wiederum auch annotiert werden können. Der Nutzer wird bei seiner Arbeit unterstützt, da Bearbeitungsschritte unterbrechbar sind und der aktuelle Stand jeweils mittels der CHAO beschrieben ist, was eine spätere Fortsetzung ermöglicht. Weiterhin ist der Aufbau modular, wodurch eine Erweiterbarkeit gegeben ist. Als Beispiel kann ein Kurator auf Basis der aufgenommenen und semantisch beschriebenen Änderungen analysieren, welche Personen welche Änderungen wann vorgenommen hat. Die Anwendung von (Co-)Evolutionsmechanismen ist in diesem Framework nicht umgesetzt. Rollen- und Zugriffsmanagement sowie Freigabeprozesse werden ebenfalls nicht beachtet.

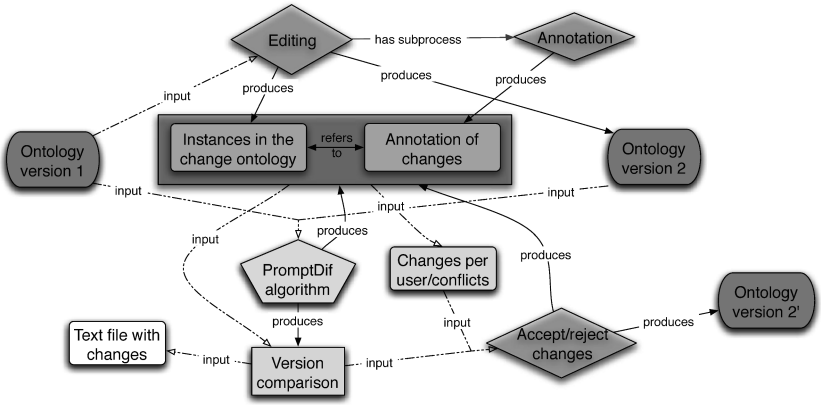


Abbildung 3.9: Komponenten des Frameworks zur Unterstützung der Ontologieevolution in Protégé [Noy+06, S. 550]

[7] https://protegewiki.stanford.edu/wiki/Changes_Tab (besucht am 29.11.2020)
[8] https://protege.stanford.edu/ (besucht am 29.11.2020)

3.2.6 R43ples

Etablierte Revisionsverwaltungssysteme, wie git, SVN, CVS, Bazaar und Mercurial, sind für die Revisionsverwaltung von Modellen nur bedingt nutzbar, da sie auf einer Zeilenbasis beim Vergleich arbeiten. Die Beschreibung von Informationsmodellen erfolgt hingegen meist in Sprachen, für die die interne Abfolge keine Rolle spielt. Beim zeilenbasierten Vergleich würden dann Änderungen erkannt werden, die keine Änderung am Informationsmodell zur Folge haben. Als Beispiele können hierfür XML, OWL und RDF aufgeführt werden. Für eine nachvollziehbare Speicherung von Änderungen an Informationsmodellen müssen daher entsprechende Möglichkeiten geschaffen werden, die diesen Herausforderungen begegnen können. Beispiele dafür sind bereits im Bereich des Semantic Webs vorzufinden. Canova *et al.* [Can+15] gibt einen Überblick zu bestehenden Ansätzen und Implementierungen. Nur zwei der verglichenen Ansätze erlauben dabei die Revisionsverwaltung in einer verteilten und semantischen Art und Weise, wie sie im Bereich des Semantic Web notwendig ist. R43ples [GHU14; GHU16] ist eines dieser beiden Systeme, wobei dieses zusätzlich eine vollständige semantische Beschreibung der Revisionsverwaltung umsetzt. Diese ist im Ansatz R&Wbase [Van+13] nicht durchgängig gegeben.

R43ples wurde in vorangegangenen Arbeiten entwickelt [Hen13; GHU14; GHU16] und ermöglicht eine Revisionsverwaltung im Semantic Web. Hierbei verfolgt es einen vollständig semantischen Ansatz für die Beschreibung der Revisionsinformation in Linked Data und basiert zu Teilen auf der Arbeit von Vander Sande *et al.* [Van+13]. Das System agiert als ein Proxy, der vor bestehende Triple Stores geschaltet werden kann, um die Revisionsverwaltungsfunktionalität den entsprechenden Triple Stores hinzuzufügen. Als Interface für die Interaktion werden erweiterte SPARQL 1.1 Funktionalitäten verwendet. Die grundlegende Architektur ist in Abbildung 3.10 dargestellt. In weiteren Arbeiten wurde R43ples bereits um erste Ansätze für die Zusammenführung von divergierten Zweigen erweitert [Hen14; HGU16], wobei derzeit keine Mechanismen zur Aggregation von Änderungen zu semantischen High-Level-Changes angewendet werden. Da es sich bei R43ples um ein reines Revisionsverwaltungssystem handelt, sind keine Mechanismen zur (Co-)Evolution umgesetzt. Ebenso besitzt es derzeit kein Rollen- oder Zugriffsmanagement. Hier kann nur auf proprietäre Lösungen der angeschlossenen Triple Stores zurückgegriffen werden. R43ples bietet neben dem bereitgestellten SPARQL-Endpoint auch eine Weboberfläche zur Bedienung an.

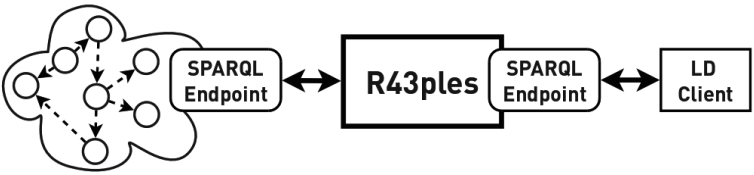


Abbildung 3.10: Grundlegende Architektur des semantischen Revisonsverwaltungssystems R43ples [GHU14, S. 5]

3.2.7 Zusammenfassung

Die in den vorangegangenen Abschnitten vorgestellten Ansätze werden in Tabelle 3.3 anhand der aufgenommenen Anforderungen gegenübergestellt. Die Bewertungskriterien teilen sich in vollständige (✓✓), teilweise (✓) oder keine Erfüllung (✗) der jeweiligen Anforderung ein.

Zusammenfassend lässt sich feststellen, dass keiner der Ansätze alle Anforderungen abdeckt. Im Bereich des Nutzungskontextes werden Zugriffe und Freigabeprozesse in keinem der Ansätze beschrieben. Diese Problematiken werden meist nur im Ausblick angesprochen. Hingegen bieten alle Ansätze ein Nutzerinterface, was den Nutzer bei seiner Arbeit unterstützt und es werden auch teilweise die beteiligen Rollen definiert. Im Bereich der Änderungsmanagements und der Evolution ist festzuhalten, dass zumeist entweder Anforderungen im Bereich der Revisionsverwaltung oder im Bereich der Evolution umgesetzt werden. Eine integrierte Nutzung von Revisionsverwaltung und Evolutionsmechanismen ist höchstens zu Teilen umgesetzt. Die Modelle werden in allen Ansätzen semantisch beschrieben und auch zusätzliche Qualitätsattribute sind entweder bereits umgesetzt oder können integriert werden. Im Bereich der Revisionsverwaltung von Modellen lässt sich ein deutlicher Fortschritt von reiner Änderungsverfolgung zu Systemen mit Revisionsverwaltungsfunktionalitäten erkennen, wie sie aus etablierten Systemen bekannt sind.

Tabelle 3.3: Spiegelung der Anforderungen an existente Ansätze

	Anforderungen		Diss. Kehrer	Diss. Stojano.	SecVolution	Simantics	Changes Tab	R43ples
Nutzungs-kontext	A-101	Selbstbeschreibungsf.	✓✓	✓✓	✓	✓✓	✓	✓
	A-102	Rollenmanagement	✓	✗	✓	✓✓	✗	✗
	A-103	Zugriffsmanagement	✗	✗	✗	✗	✗	✗
	A-104	Freigabeprozesse	✗	✗	✗	✗	✗	✗
Änderungs-management	A-201	Nachvollziehbarkeit	✗	✓	✗	✓	✓	✓✓
	A-202	Zusammenführung	✓	✗	✗	✗	✗	✓
	A-203	High-Level-Changes	✓✓	✓	✓	✗	✓	✗
	A-204	Revisionsmodell	✗	✓✓	✗	✗	✓✓	✓✓
Evolution	A-301	Strategien	✓	✓	✓	✗	✗	✗
	A-302	Sem. Beschr.	✗	✗	✓	✗	✗	✗
	A-303	Vollständigkeitsp.	✓	✓	✓	✗	✗	✗
Sem. Modellbeschr.	A-401	Rev. Modelle	✓✓	✓	✓✓	✓	✓✓	✓
	A-402	Verbindungen	✗	✓	✗	✓	✗	✗
Qualitäts-attribute	A-501	Berechnung	✓	✓✓	✓	✓	✓	✓
	A-502	Integration	✓	✓	✓	✓	✓	✗

3.3 Analyseergebnisse und Priorisierung

Die vorangegangene Analyse zeigt die notwendigen Anforderungen an ein System, um mit Änderungen in einer kollaborativen Umgebung umgehen zu können und sowohl die Änderungen nachvollziehbar zu dokumentieren wie auch die automatische Evolution zu unterstützen. Es wird ersichtlich, dass nur eine integrierte Lösung aus Revisionsverwaltungsfunktionalitäten und den Evolutionsmechanismen diese Anforderungen erfüllen kann. Hierdurch können Nutzer bestmöglich bei ihrer Arbeit unterstützt werden und die Möglichkeit, dass Fehler auftreten, wird verringert. Die Herausforderungen, die durch die beiden Anwendungsfälle beschrieben werden, bestätigen die allgemein aufgenommenen Anforderungen an ein solches System. Durch die Spiegelung der Anforderungen an existente Ansätze wird des Weiteren die Notwendigkeit eines solches RMS nachgewiesen, da keines der bestehenden Systeme alle Anforderungen erfüllt und die Integration von Revisionsverwaltung und Evolution bisher nur wenig Betrachtung findet.

Für die Umsetzung eines RMS muss im ersten Schritt ein Rahmen geschaffen werden, in dem die einzelnen Anforderungen umgesetzt werden können. Die zentrale Anforderung ist die semantische Modellbeschreibung auf deren Basis wiederum weiterführende Anforderungen umgesetzt werden können. So kann das Änderungsmanagement darauf agieren, um Basisrevisionskontrollfunktionalitäten sowie weiterführende Mechanismen, wie die semantische Aggregation zu High-Level-Changes und die Zusammenführung von divergierten Entwicklungszweigen, anzubieten. Die Beschreibung der durchgeführten Aktionen erfolgt dabei ebenfalls semantisch. Für die Etablierung von Evolutionsmechanismen spielt wiederum das Änderungsmanagement eine große Rolle, da auf dessen Basis Änderungen detektiert und notwendige Evolutionen abgeleitet werden können. Durchgeführte Evolutionsschritte können dann im Änderungsmanagement semantisch beschrieben abgelegt werden. Alle weiteren Anforderungen, wie der Nutzungskontext und weitere Qualitätsattribute, sind zusätzliche Funktionalitäten. Diese müssen ebenfalls für die Realisierung eines RMS, das allen aufgeführten Anforderungen genügt, umgesetzt werden. Im weiteren Verlauf dieser Arbeit spielen diese zusätzliche Funktionalitäten daher eine untergeordnete Rolle, da sich auf die grundlegenden Mechanismen eines RMS beschränkt wird und die Zusatzfunktionen oft auch an eine technische Umsetzung gekoppelt sind. Die grundlegenden Beschreibungen werden im Folgenden jedoch technologieunabhängig formuliert, um ein größtmögliches Einsatzspektrum und die Übertragbarkeit auf unterschiedliche Einsatzszenarien zu ermöglichen.

4 Entwurf

Auf Basis der erhobenen Anforderungen wird in diesem Abschnitt ausgehend vom Lebenszyklus von Informationsmodellen ein Konzept für ein RMS vorgestellt. Mittels eines Komponentendiagramms werden die notwendigen Komponenten und deren Interaktionspunkte dargestellt. Im Folgenden werden, die für die einzelnen Komponenten notwendigen mathematischen Definitionen und semantischen Beschreibungen technologieunabhängig beschrieben, was wiederum die Grundlage für notwendige Algorithmen der Umsetzung bildet.

4.1 Lebenszyklusmodell für Informationsmodelle

Informationsmodelle unterliegen, wie auch Software oder Produkte im Allgemeinen, einem Lebenszyklus. Aus der Softwareentwicklung heraus sind bekannte Vertreter das Wasserfallmodell oder agile Softwareentwicklungsmodelle. Ebenso stellt die DIN EN 62890 [DIN17] den Lebenszyklus von Produkten und Systemen dar, wobei hierbei der Fokus auf der Mess-, Steuer- und Regelungstechnik im industriellen Umfeld liegt. Daraus ist erkennbar, dass es sich bei Lebenszyklusmodellen meist um domänenspezifische Modelle handelt. Im Bereich der Informationsmodelle existieren ebenso Ansätze wie unter anderem im Bereich von Goverment Linked Data (GLD) [W3C12], Linked Open Data (LOD) [Aue+12] oder dem Ontologie- beziehungsweise Wissensmanagements [Sur+08; SSS04].

Abbildung 4.1 zeigt ein aus den bestehenden Ansätzen heraus abgeleitetes verallgemeinertes Lebenszyklusmodell für Informationsmodelle, das bereits in [HGU18] veröffentlicht ist. Dieses bildet auf der einen Seite die durchlebbaren Phasen eines Informationsmodells ab. Auf der anderen Seite stellt es aber auch die Elemente der Revisionsverwaltung und der Sicherheit dar, die ein Informationsmodell über den gesamten Lebenszyklus hinweg begleiten.

Der Lebenszyklus beginnt, wie auch in der Software- oder Produktentwicklung, mit einer Erhebung von Anforderungen, die in der anschließenden Erstellungsphase entsprechend umgesetzt werden. Es folgt die Phase der Veröffentlichung, durch die die Modelle für weitere Anwendungen nutzbar werden und beispielsweise Datenmodelle auf Basis eines veröffentlichten Informationsmodells erzeugt werden können. In der Phase der Nutzung wird das Informationsmodell produktiv eingesetzt, wobei aufgrund der Nutzung oder auch durch äußere Einflüsse Anforderungsänderungen beziehungsweise Erweiterungsbedarf entsteht. Beispiele hierfür sind unter anderem in [HKB17] für das UML-Metamodell oder BPMN aufgeführt. Die notwendigen Änderungen resultieren dann in einer Evolution des Informationsmodells, um eine semantische Erosion von bestehenden Definitionen vorzubeugen. Der Evolutionsprozess kann sich in der Nutzungszeit mehrfach wiederholen. Die Außerdienstsetzung ist die letzte Phase im Lebenszyklus eines Informationsmodells und kann sich beispielsweise durch eine Archivierung der Daten-

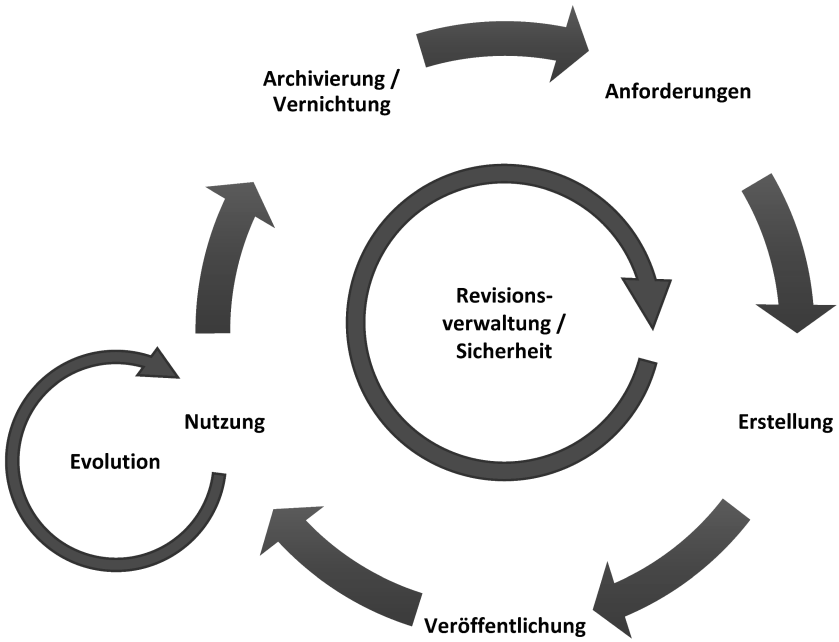


Abbildung 4.1: Allgemeines Lebenszyklusmodell von Informationsmodellen (veröffentlicht in [HGU18, S. 49])

und Informationsmodelle oder eine vollständige Vernichtung der Inhalte auszeichnen. In der Praxis werden Informationsmodelle jedoch vorrangig weiterentwickelt und an die sich ändernden Anforderungen angepasst. Eine vollständige Außerdienstsetzung wird nur im Zusammenhang mit sehr disruptiven Änderungen der Anforderungen eintreten. In diesem Fall wird dann auf Basis der neuen Anforderungen ein neues Informationsmodell entwickelt, wobei die Erfahrungen aus dem alten Modell in den meisten Fällen einfließen.

Sowohl die Revisionsverwaltung als auch die Sicherheit der Informationsmodelle sind während aller Phasen von zentraler Bedeutung. So werden durch eine Revisionsverwaltung Änderungen an den Modellen stets nachvollziehbar gespeichert, was die Möglichkeit des Rücksprungs auf vorangehende Versionsstände ermöglicht. Ebenso können verschiedene Versionen und Varianten gepflegt und kollaborativ weiterentwickelt werden. In diesem Zusammenhang und auch während der Nutzung eines Modells spielt die Zugriffssicherheit eine wichtige Rolle. So müssen geeignete Maßnahmen ergriffen werden, um den unerlaubten Zugriff auf beispielsweise abgeleitete Datenmodelle oder die zugehörigen Informationsmodelle einzuschränken. Dies ist notwendig, da in diesen sensible Kundendaten oder das eigene Know-how enthalten sein können.

4.2 Revision Management System

Die Umsetzung der aufgenommenen Anforderungen erfordert, wie bereits in Abschnitt 3.3 aufgeführt, die Konzeption eines Systems, das sowohl Änderungen nachvollziehbar speichern kann, als auch die Evolution von Informations- und Datenmodellen unterstützt. Dieses System wird im Folgenden als RMS bezeichnet, da es über eine reine Revisionsverwaltung hinausgeht und erweiterte Funktionen für den Umgang mit Anforderungsänderungen in Informations- und Datenmodellen anbietet.

4.2.1 Komponentenübersicht

Das RMS besteht im Wesentlichen aus drei Komponenten, deren Aufteilung an das Model-View-Controller-Prinzip aus der Softwareentwicklung angelehnt ist. Abbildung 4.2 zeigt die resultierenden grundlegenden Zusammenhänge zwischen den Komponenten. Eine vollständige Darstellung aller Zusammenhänge ist in Abbildung A.1 dargestellt.

Die Komponente *DataManagement* besteht aus dem eigentlichen Datenspeicher und einem Revisionskontrollsystem, das auf diesen Datenspeicher zugreift und alle Änderungen nachvollziehbar semantisch beschreibt. Durch diese Komponente werden die unter A-200 zusammengefassten Anforderungen umgesetzt. Das Revisionskontrollsystem schließt dabei sowohl die Basisrevisionskontrollfunktionalitäten wie die Erstellung von neuen Entwicklungszweigen, neuen Tags und neuen Commits, als auch erweiterte Funktionen, wie die Zusammenführung divergierter Entwicklungszweige und die semantische Aggregation von Änderungen, ein. Innerhalb von *DataManagement* können sowohl Daten- als auch Informationsmodelle revisioniert werden, was der Umsetzung von Anforderung A-401 entspricht. *Control* übernimmt auf Basis der Funktionalitäten von *DataManagement* den Umgang mit den Änderungen in Bezug auf die Evolution von verbundenen Modellen, was der Realisierung der unter A-300 kategorisierten Anforderungen entspricht. Die zugrunde liegenden Verbindungen (siehe Anforderung A-402), auf deren Basis die Evolution ausgeführt werden kann, werden durch den *ConnectionManager* beschrieben, wobei dessen Datenhaushalt auch im *DataManagement* vorgehalten wird. Zur Einschränkung von Zugriffsrechten auf die einzelnen Komponenten existiert des Weiteren der *PermissionAndApprovalProcessManager*, hiermit können die Anforderungen A-102 und A-103 umgesetzt werden. Mit dessen Hilfe können außerdem Freigabeprozesse (Anforderung A-104) realisiert werden. Das *UserInterface* ermöglicht eine grafische Nutzung der Funktionalitäten des RMS und stellt damit die Grundlage für die Realisierung von Anforderung A-101 bereit. Zusätzliche Qualitätsattribute, wie in der Anforderungssammlung A-500 gefordert, können auf Basis des Gesamtsystems und der bereitgestellten semantischen Beschreibung realisiert werden.

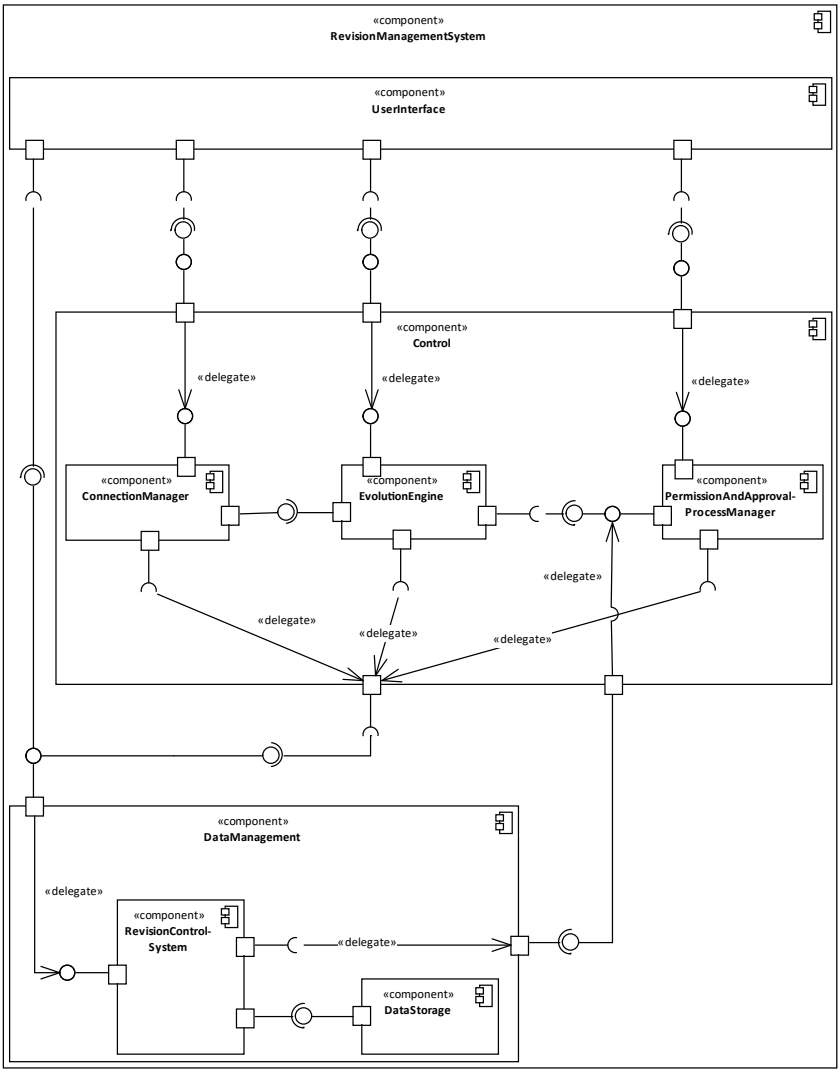


Abbildung 4.2: UML-Komponentendiagramm des RMS

4.2.2 Data Management

Die *DataManagement*-Komponente setzt sich in der ersten Ebene aus zwei Komponenten zusammen, wie in Abbildung 4.3 dargestellt. Hierzu gehören der *DataStorage*, der den gesamten Datenhaushalt des RMS vorhält, und das *RevisionControlSystem*, das die vollständige Revisionierung der Daten vornimmt. Diese Komponente ist wiederum untergliedert in mehre Teilkomponenten, die die Basisrevisionskontrollfunktionalitäten (*BasicRevisionControl*, Anforderung A-201), die Zusammenführung von divergierten Entwicklungszweigen (*MergeManagement*, Anforderung A-202) und die Aggregation von Änderungen auf eine semantisch nachvollziehbare Ebene (*HighLevelChangeAggregation*, Anforderung A-203) realisieren. Die Basisfunktionalitäten der Revisionsverwaltung bilden dabei die Grundlage für erweiterte Funktionalitäten wie die Zusammenführung von divergierten Entwicklungszweigen und die Umsetzung von Aggregationsmechanismen. Die Zusammenführung erfordert dabei einerseits die Analyse der Historie der zusammenzuführenden Zweige und andererseits muss das Ergebnis der Zusammenführung wiederum semantisch beschrieben abgelegt werden. Gleiches gilt für die Aggregation, um Änderungen über mehrere Revisionen hinweg nachzuvollziehen und semantisch zu aggregieren. Dieser Mechanismus benötigt einen Regelsatz, auf dessen Basis die Aggregation vollzogen werden kann. Dieser wird durch den *HLCAggRuleManager* bereitgestellt, wobei der zugrunde liegende Datenhaushalt ebenfalls revisioniert abgelegt werden kann. Die *HLCAggMechanism*-Komponente übernimmt die Auswertung der Regelsätze und wendet diese auf die zu analysierenden Revisionsinformation an. Die Ergebnisse dieser Analyse können entweder direkt im Datenspeicher semantisch und revisioniert abgelegt oder anderen Komponenten zur Weiterverarbeitung zur Verfügung gestellt werden. Innerhalb des *DataManagement*s können diese für die Zusammenführung von Entwicklungszweigen herangezogen werden, um beispielsweise eine detaillierte Konfliktanalyse beziehungsweise -behebung vornehmen zu können.

Nach außen stellt die *DataManagement*-Komponente Schnittstellen für den revisionssicheren Zugriff auf den Datenhaushalt, die Zusammenführung von Entwicklungszweigen und die Aggregation von Änderungen bereit. Benötigt wird eine Schnittstelle an ein Autorisierungssystem, um einen Zugriffsschutz auf die gespeicherten Daten umsetzen zu können.

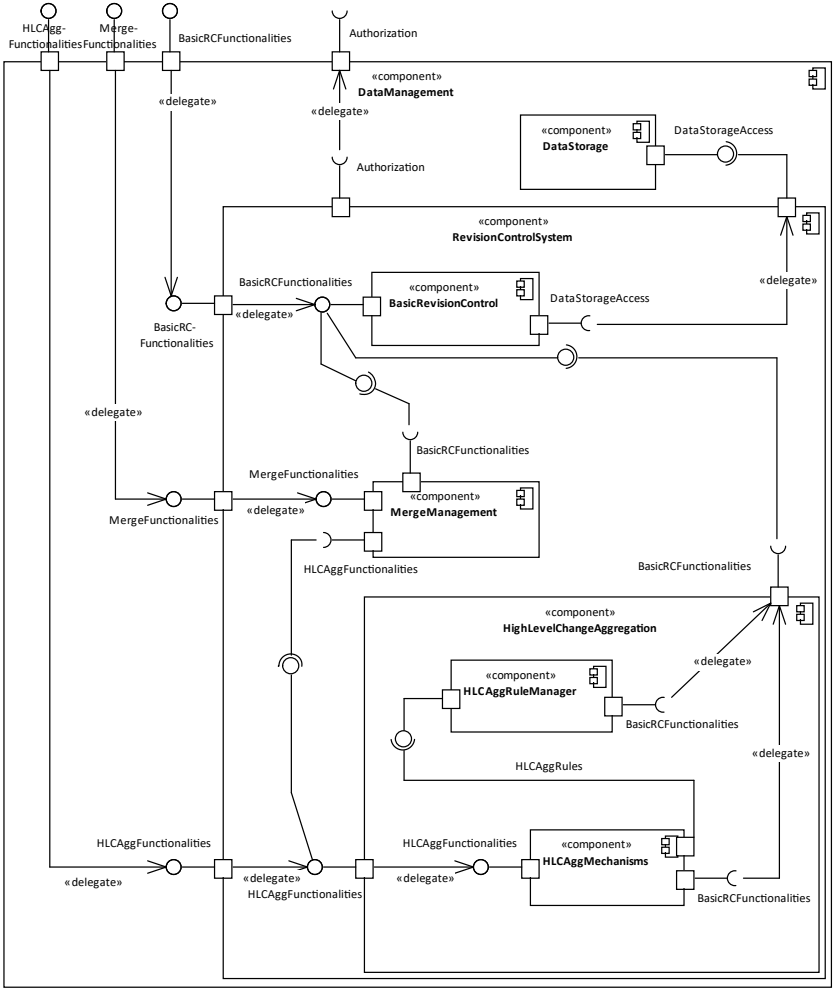


Abbildung 4.3: Ausschnitt der *DataManagement*-Komponente aus dem RMS

4.2.3 Control

In der ersten Ebene setzt sich die *Control*-Komponente, dargestellt in Abbildung 4.4, aus drei Teilkomponenten zusammen. Die *EvolutionEngine* (Anforderung A-301) ist dabei für die Umsetzung von Co-Evolutionen mittels der *CoEvolutionMechanisms*-Komponente verantwortlich. Für die Ausführung wird wiederum ein Regelsatz benötigt, der durch eine entsprechende Komponente (*EvoRuleManager)* vorgehalten wird. Zusätzlich werden Verbindungen innerhalb und zwischen Modellen durch eine separate Komponente *ConnectionManager* (Anforderung A-402) bereitgestellt und zugreifbar gemacht, wodurch sie der *EvolutionEngine* zur Verfügung stehen und für die Co-Evolutionen als Grundlage herangezogen werden können. Regelsätze und die Ergebnisse der Co-Evolutionen werden revisonssicher und semantisch beschrieben im *DataManagement* abgelegt. Die *EvolutionEngine* hat des Weiteren Zugriff auf alle weiteren angebotenen Schnittstellen des *DataManagement*s, um beispielsweise bei Bedarf Entwicklungszweige automatisiert zusammenzuführen oder die aggregierten Änderungen als Grundlage für die Anwendung der Regelsätze zu nutzen. Zugriffsbeschränkungen im RMS werden mittels des *PermissionAndApprovalProcessManager*s realisiert. Dieser besitzt einen *PermissionManager*, der für die Authentifizierung und Autorisierung von Nutzern verantwortlich ist. Dafür wird eine Nutzerverwaltung benötigt, die entweder intern vorgehalten wird oder über eine weitere Schnittstelle eingebunden werden kann. Weiterhin ist eine *ApprovalProcessManager*-Komponente integriert, die die Realisierung von Freigabeprozessen im RMS verwaltet. So können in dieser Komponente Abläufe geplant und entsprechende verantwortliche Nutzerrollen zugeordnet werden. Hierfür wird wiederum Zugriff auf den Datenspeicher benötigt, um zum Beispiel Vorgehens- und Reviewprozesse für die Zusammenführung von divergierten Entwicklungszweigen zu definieren.

Die *Control*-Komponente stellt nach außen Schnittstellen für Autorisierung, Authentifizierung, Nutzerrollenverwaltung, Freigabeprozessmanagement sowie für die Verwaltung von Verbindungen innerhalb und zwischen Modellen sowie für die Durchführung von Co-Evolutionen zur Verfügung. Für die Realisierung dieser Schnittstellen werden Schnittstellen auf die Funktionen und Daten des *DataManagement*s benötigt.

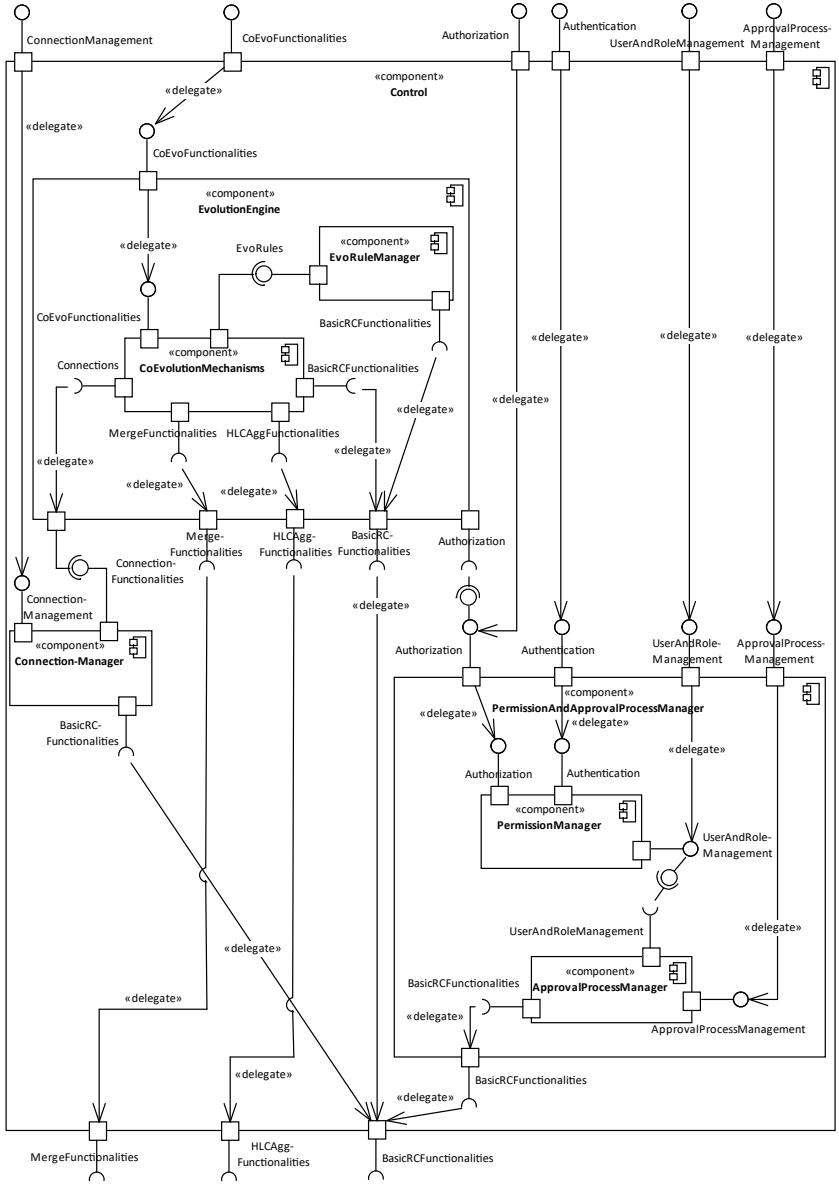


Abbildung 4.4: Ausschnitt der *Control*-Komponente aus dem RMS

4.2.4 User Interface

Das *UserInterface*, das in Abbildung 4.5 dargestellt ist, dient zur grafischen Konfiguration des RMS. Für den Zugriff auf die RMS-Funktionalitäten muss sich ein Nutzer im ersten Schritt authentifizieren. Je nach Nutzerrolle sind dann unterschiedliche Interaktionen mit dem System möglich. Hierbei sind unterschiedliche Eingriffsarten und -tiefen für die Nutzer vorgesehen. So ist es auf der einen Seite möglich, die Basisrevisionskontrollfunktionalitäten zu nutzen, um beispielsweise Änderungen an Modellen an das System zu übertragen oder Entwicklungszweige zusammenzuführen. Auf der anderen Seite können aber auch auf einer abstrakteren Stufe Änderungen an den Modellen vorgenommen werden, um zum Beispiel Verbindungen innerhalb oder zwischen Modellen grafisch zu editieren. Ebenso können vorgenommene Änderungen an abhängige Modelle co-evolviert werden. Je nach Rechten des Nutzers können ebenso die Nutzer und Rollen verwaltet oder Freigabeprozesse definiert oder verändert werden. Durch eine grafische Repräsentation des Systems kann der Nutzer das System zum einen an die gewünschten Anforderungen anpassen und zum anderen automatisiert durchgeführte Aktionen überprüfen und bestätigen oder rückgängig machen.

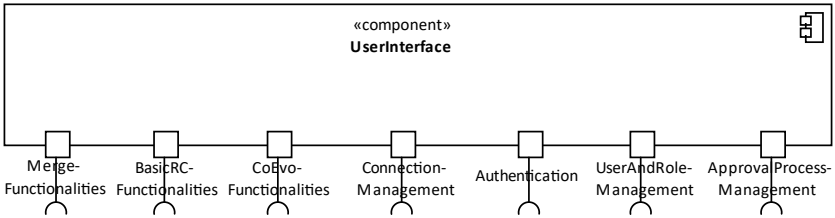


Abbildung 4.5: Ausschnitt der *UserInterface*-Komponente aus dem RMS

4.3 Formale Beschreibung verbindungsorientierter Modelle

Die Analyseergebnisse haben aufgezeigt, dass eine semantische Modellbeschreibung vorliegen muss. Hierzu gehören einerseits Informations- und Metamodelle (Anforderung A-401), um die Semantik des jeweiligen Modells zu beschreiben, andererseits aber auch die Relationen innerhalb eines Modells oder zwischen unterschiedlichen Modellen (Anforderung A-402). Die resultierenden Beschreibungen werden innerhalb des RMS in der *ConnectionManager*-Komponente vorgehalten. Bestehende Ansätze fokussieren sich entweder vorwiegend auf formale Beschreibungen in Richtung von UML-Klassendiagrammen oder in Richtung von Diagrammen, die die Darstellung von Komponenten- oder Zustandsdiagrammen erlauben. Beispiele hierfür sind unter anderem [Keh15], [Tap99] und [FMP99]. Im Folgenden wird eine abstrahierte formale Beschreibung auf Basis von Compound Graphs eingeführt, die eine Übertragung der darauf aufbauenden Beschreibungen auf unterschiedliche Anwendungsbereiche ermöglicht.

4.3.1 Compound Graphs

Compound Graphs werden vorwiegend unbemerkt in unterschiedlichen Domänen verwendet, wie beispielsweise in UML-Diagrammen (zum Beispiel UML-Zustandsdiagrammen) oder Schaltkreisdiagrammen. In der Literatur wird diese Art von Graphen vorrangig für den Zweck des automatischen Layouts der visuellen Repräsentation von Diagrammen genutzt. Darüber hinaus eignet sich diese auch für die formale Beschreibung von Zusammenhängen in Informations- und Datenmodellen. Nachfolgend werden hierzu die grundlegenden formalen Definitionen auf Basis von Sander [San05] dargestellt.

Ein Compound Graph $\tilde{G}$ besteht aus einem einfach gerichteten Graphen $\tilde{G}'$ und einem Baum $\tilde{T}'$, wie in Gleichung 4.1 dargestellt.

$$\tilde{G} = (\tilde{G}', \tilde{T}') \tag{4.1}$$

Allgemein gilt, dass ein einfach gerichteter Graph $\hat{G}$ aus einer Menge an Knoten $\hat{V}$ und einer Menge an Kanten $\hat{E}$ besteht, wobei gilt, dass $\hat{E} \subseteq \hat{V} \times \hat{V}$. Zusammengefasst lässt sich $\hat{G}$ durch $\hat{G} = (\hat{V}, \hat{E})$ beschreiben. Vorgänger und Nachfolger können in $\hat{G}$ mittels den Funktionen in Gleichung 4.2 ermittelt werden. Des Weiteren gilt die in Gleichung 4.3 dargestellte Symbolik.

$$\begin{aligned} \boldsymbol{pred}_{\hat{G}}(\hat{v}) &:= \left\{ \hat{w} \in \hat{V} \;\middle|\; (\hat{w}, \hat{v}) \in \hat{E} \right\} \\ \boldsymbol{succ}_{\hat{G}}(\hat{v}) &:= \left\{ \hat{w} \in \hat{V} \;\middle|\; (\hat{v}, \hat{w}) \in \hat{E} \right\} \end{aligned} \tag{4.2}$$

$$\begin{aligned} &\hat{v} \underset{\hat{G}}{\rightarrow} \hat{w} \ldots \; Kante\,(\hat{v}, \hat{w}) \in \hat{E} \\ &\hat{v} \underset{\hat{G}}{\rightarrow}^{*} \hat{w} \ldots \; Sequenz\ von\ Kanten/Pfad\ (eventuell\ leer) \\ &\hat{v} \underset{\hat{G}}{\rightarrow}^{*} \hat{v} \ldots \; wenn\ nicht\ leer, dann\ Zyklus, ansonsten\ azyklisch \end{aligned} \tag{4.3}$$

Bei einem Baum $\hat{T}$ handelt es sich allgemein um einen azyklischen gerichteten Graphen mit $\hat{T} = (\hat{V}, \hat{E})$. Dabei gilt, dass $\hat{T}$ aus $\hat{n}$ Knoten und $\hat{n} - 1$ Kanten besteht. Des Weiteren existiert ein Wurzelknoten, für den Gleichung 4.4 erfüllt ist. Blätter innerhalb des Baums besitzen keine Nachfolgeknoten, daher gilt, dass $\boldsymbol{succ}_{\hat{G}}(\hat{v}) = \emptyset$. Alle anderen Knoten werden als innere Knoten bezeichnet.

$$\hat{r} \underset{\hat{T}}{\rightarrow}^{*} \hat{v},\ für\ alle\ \hat{v} \in \hat{V} \tag{4.4}$$

Auf Basis der vorangegangenen allgemeinen Definitionen kann im Folgenden die Definition von Compound Graphs aus Gleichung 4.1 weiter detailliert werden. Die zugehörige Definition wird in Gleichung 4.5 gegeben. Hierbei besteht die Menge der Knoten $\tilde{V}$ aus der Menge der Basisknoten $\tilde{B}$ (Blättern von $\tilde{T}'$, $\tilde{B} = \{\tilde{b} \in \tilde{V} | \boldsymbol{succ}_{\tilde{G}}(\tilde{b}) =$

$\emptyset\}$) und der Menge der Subgraphen $\tilde{S}$ (innere Knoten von $\tilde{T}'$, $\tilde{S} = \{\tilde{s} \in \tilde{V} | \boldsymbol{succ}_{\tilde{\boldsymbol{G}}}(\tilde{s}) \neq \emptyset\}$). Daher gilt, dass $\tilde{V} = \tilde{B} \cup \tilde{S}$.

Die Verbindungsrelationen zwischen Basisknoten und Subgraphen werden mittels $\tilde{G}'$ beschrieben. Resultierende Kanten $\tilde{E}_{\tilde{G}'}$ werden auch als Adjazenzkanten bezeichnet. Zusätzlich repräsentiert $\tilde{T}'$ Verschachtelungsbeziehungen. So können Subgraphen wiederum andere Subgraphen enthalten, was mit den Kanten $\tilde{E}_{\tilde{T}'}$ beschrieben wird, die auch als Hierarchie- oder Inklusionskanten bezeichnet werden.

$$\begin{aligned} \tilde{G}' &= (\tilde{B} \cup \tilde{S}, \tilde{E}_{\tilde{G}'}) \\ \tilde{T}' &= (\tilde{B} \cup \tilde{S}, \tilde{E}_{\tilde{T}'}) \end{aligned} \tag{4.5}$$

Zur Visualisierung der vorangegangenen formalen Definitionen ist in Abbildung 4.6 ein Beispiel aus [San05] aufgeführt, das die Verwendung von $\tilde{G}'$ und $\tilde{T}'$ anschaulich darstellt. G' beziehungsweise T' in der Abbildung entsprechen dabei den hier aufgeführten Definitionen $\tilde{G}'$ beziehungsweise $\tilde{T}'$.

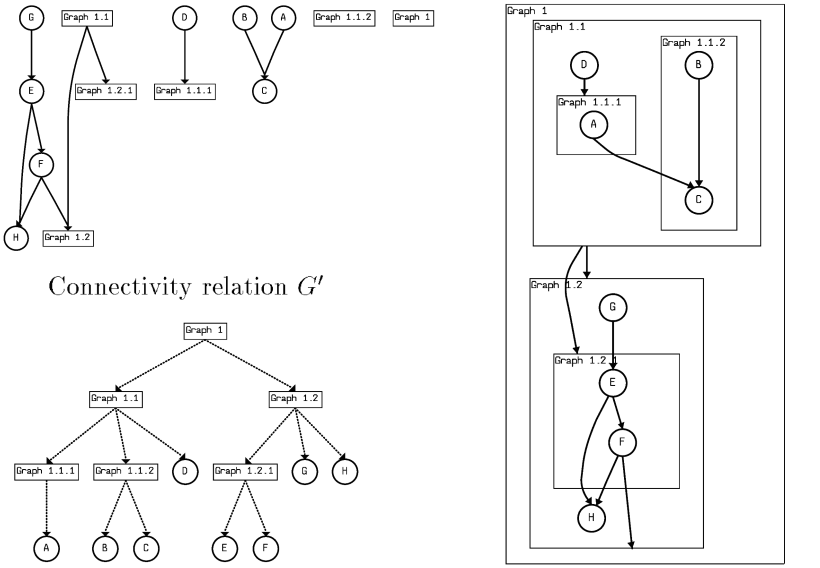


Abbildung 4.6: Beispiel für die Nutzung von Compound Graphs [San05, S. 3]

4.3.2 Compound Graphs Erweiterung

Compound Graphs bieten durch die Vereinigung von unterschiedlichen Strukturierungsmöglichkeiten, wie Hierarchiebildung, Relationsbeschreibung aber auch Gruppierungen, eine gute Grundlage für unterschiedliche Anwendungsbereiche. In den einzelnen Anwendungsbereichen werden diese Strukturierungsmöglichkeiten jedoch unterschiedlich verwendet, was dazu führt, dass für die Spezialisierung der entsprechenden Ausprägung eine Semantik zuordenbar sein muss. Abbildung 4.7 zeigt exemplarische Umsetzungsmöglichkeiten auf Basis von Compound Graphs in unterschiedlichen Anwendungsbereichen. Die Knoten können dabei unterschiedliche Interpretationen annehmen, wie zum Beispiel (Teil-)Modelle, Typenmodell, komplexe Objekte, Gruppen, Software Packages, Komponenten oder auch Attribute, Ports und Zustände. Durch Adjazenzkanten können unter anderem Typbeziehungen, Objektrelationen oder einfache Verbundenheitsbeziehungen und allgemeine Relationen ausgedrückt werden. Inklusionskanten wiederum ermöglichen die Abbildung von hierarchischen Zusammenhängen, wie Beinhaltet- oder Vererbungsbeziehungen.

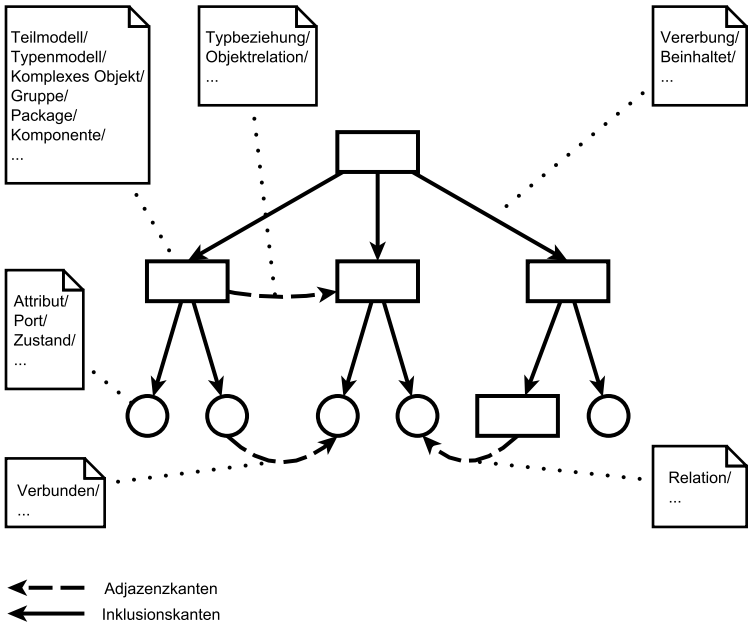


Abbildung 4.7: Beispiele für Umsetzungsmöglichkeiten mit Compound Graphs

Zur Beschreibung der Semantik müssen sowohl die Knoten als auch die Kanten um ein entsprechendes Element erweitert werden. Dies resultiert in den Definitionen, die in Gleichung 4.6 aufgeführt sind. Ein Knoten wird dadurch zu einem Tupel, bestehend aus einem Identifikator $\tilde{n}_v$ und einem Identifikator $\tilde{n}_z$, der die Semantik zuordnet. Diese beiden sind jeweils aus der Menge der natürlichen Zahlen größer Null ($\mathbb{N}^+$). Die natürlichen Zahlen werden hierbei zur vereinfachten Darstellung und Berechnung von neuen Identifikatoren verwendet und können bei einer Umsetzung durch ein beliebiges anderes eindeutiges Identifikationsschema beziehungsweise die Definitionen aus [Int+19] ersetzt werden. Für die Kanten von $\tilde{G}'$ und $\tilde{T}'$ gilt ebenso, dass die Semantik über ein weiteres Element $\tilde{n}_z$ aus $\mathbb{N}^+$ zugeordnet wird, woraus sich für die Kanten jeweils Tripel ergeben. Die vorangegangenen Definitionen aus Gleichung 4.2 bleiben bestehen, agieren jedoch unabhängig von der Semantik der Kanten.

$$\begin{aligned}
\tilde{V} &:= \left\{(\tilde{n}_v,\tilde{n}_z) \;\middle|\; \tilde{n}_v \in \mathbb{N}^+ \wedge \tilde{n}_z \in \mathbb{N}^+\right\} \\
\tilde{E}_{\tilde{G}'} &:= \left\{(\tilde{v},\tilde{w},\tilde{n}_z) \;\middle|\; \tilde{v} \in \tilde{V} \wedge \tilde{w} \in \tilde{V} \wedge \tilde{n}_z \in \mathbb{N}^+\right\} \\
\tilde{E}_{\tilde{T}'} &:= \left\{(\tilde{v},\tilde{w},\tilde{n}_z) \;\middle|\; \tilde{v} \in \tilde{V} \wedge \tilde{w} \in \tilde{V} \wedge \tilde{n}_z \in \mathbb{N}^+\right\}
\end{aligned} \tag{4.6}$$

4.3.3 Semantische Beschreibung

Die in den Abschnitten 4.3.1 und 4.3.2 eingeführten formalen Beschreibungen lassen sich ebenso in einem UML-Diagramm erfassen, wie in Abbildung 4.8 dargestellt. Die in den vorangegangenen Abschnitten definierten Einschränkungen gelten dabei weiter, sind jedoch nicht vollständig im Modell abgebildet. Ein *CompoundGraph* besteht aus einer Menge an Knoten (*Node*) und einer Menge an Relationen (*Relation*). Relationen bilden dabei die Verbindungen zwischen Knoten ab. Da diese in Compound Graphs gerichtet sind, wird ein Start- (*start*) und ein Endknoten (*target*) zugeordnet. Die Relation wird des Weiteren in die *ConnectivityRelation* und die *NestingRelation* spezialisiert. Für die Knoten wird ebenfalls eine Spezialisierung, nach der Erweiterung aus Abschnitt 4.3.2, in *Leaf* und *Subgraph* vorgenommen.

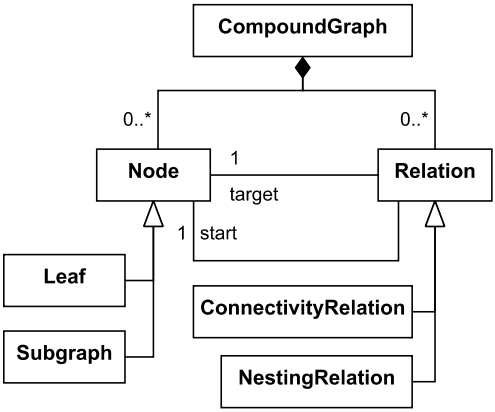


Abbildung 4.8: Semantische Beschreibung von Compound Graphs als UML-Modell

4.4 Änderungsmanagement

In diesem Abschnitt erfolgt die mathematische und semantische Beschreibung eines technologieunabhängigen delta-basierten Revisionskontrollsystems. Dieses ist unabhängig von internen Zeilenordnungen, was insbesondere eine Revisionierung von graphenbasierten Modellen ermöglicht. Diese Beschreibungen sind notwendig, da sie die Grundlage für alle weiteren Konzepte bilden, die auf die Revisionskomponente (*BasicRevisionControl*) zugreifen und auf Basis dieser agieren. Die nachfolgenden Definitionen heben dabei die in [AM17] vorgestellten Konzepte auf ein generelles Level, was eine technologieunabhängige Beschreibung ermöglicht, die dann wiederum für unterschiedliche Technologien technologiespezifisch angewendet werden kann. Des Weiteren werden die Basisrevisionskontrollfunktionalitäten (*BasicRevisionControl*) um die semantische Aggregation der Änderungen (*HighLevelChangeAggregation*) und die Zusammenführung von divergierten Entwicklungszweigen (*MergeManagement*) erweitert.

Für die semantische Beschreibung werden UML-Modelle verwendet, die über die folgenden Abschnitte kontinuierlich erweitert werden. Elemente, die aus anderen UML-Modellen erweitert oder referenziert werden, sind dabei jeweils grau markiert. Bestehende Assoziationen und Aggregationen werden in neuen UML-Modellen zumeist nicht wiederholt, sondern nur um neu hinzugekommene erweitert.

Da sich die nachfolgenden Beschreibungen vorrangig auf die Revisionierung von graphenbasierten Modellen bezieht, wird davon ausgegangenen, dass der zu revisionierende Modellinhalt jeweils aus einer Menge an Elementen besteht, die nicht weiter zerlegt werden können und innerhalb des Modells eindeutig sind. Die daraus resultierende Menge aller möglichen Elemente wird im Folgenden mit $\mathcal{S}$ bezeichnet.

4.4.1 Revisionskontrolle

Durch die im Folgenden beschrieben Funktionalitäten (Anforderung A-201) werden die Basisfunktionen eines Revisionskontrollsystems, wie die Erstellung von neuen Commits, Tags und Entwicklungszweigen, bereitgestellt. Grundlage bilden dafür die in [AM17] vorgestellten Konzepte, die auf ein generelles technologieunabhängiges Level gehoben werden.

4.4.1.1 Revisionsgraph

Ein Revisionsgraph beschreibt die gesamte Historie der angewendeten Änderungen. Mathematisch lässt sich dieser als ein Quintupel $\mathcal{G} = (R_g,C_g,B_g,T_g,n_g)$ beschreiben. Die zugehörigen Definitionen der Einzelelemente R_g, C_g, B_g, T_g und n_g werden in Gleichung 4.7 dargestellt. Dabei wird durch den Index g die Zugehörigkeit einer Menge zu einem spezifischen Revisionsgraphen $\mathcal{G}$ gekennzeichnet.

$$
\begin{aligned}
&\mathcal{R} \subseteq \mathbb{N}^+ \\
&\mathcal{C} := \left\{ \left(r_x,r_y,C^+,C^-\right) \;\middle|\; \begin{array}{l} r_x \in \mathcal{R} \cup \{0\} \;\wedge\; r_y \in \mathcal{R} \;\wedge\; r_x \neq r_y \;\wedge \\ C^+ \subseteq \mathcal{S} \;\wedge\; C^- \subseteq \mathcal{S} \;\wedge\; C^+ \cap\; C^- = \emptyset \;\wedge \\ C^+ \cup\; C^- \neq \emptyset \end{array} \right\} \\
&\mathcal{B} := \left\{ (R_b,r_l,\Upsilon_l,n_b) \;\middle|\; R_b \subseteq \mathcal{R} \;\wedge\; r_l \in R_b \;\wedge\; \Upsilon_l \;\subseteq\; \mathcal{S} \;\wedge\; n_b \in \mathbb{N}^+ \right\} \\
&\mathcal{T} := \left\{ (r_t,\Upsilon_t,n_t) \;\middle|\; r_t \in\; \mathcal{R} \;\wedge\; \Upsilon_t \;\subseteq\; \mathcal{S} \;\wedge\; n_t \in \mathbb{N}^+ \right\} \\
&R_g \subseteq \mathcal{R},\; C_g \subseteq \mathcal{C},\; B_g \subseteq \mathcal{B},\; T_g \subseteq \mathcal{T},\; n_g \in \mathbb{N}^+
\end{aligned}
\tag{4.7}
$$

Mittels des Symbols R_g wird die Menge aller Revisionen in einem Revisionsgraphen beschrieben. Es handelt sich dabei um eine Teilmenge der natürlichen Zahlen größer Null, die im Folgenden mit $\mathbb{N}^+$ bezeichnet wird. Die Verwendung der natürlichen Zahlen dient der vereinfachten Darstellung und Berechnung von neuen Identifikatoren, wobei stets eine eineindeutige Zuordnung von Revisionsinhalt und Revisionsidentifikator gewährleistet ist. Die natürlichen Zahlen können durch beliebige andere eindeutige Identifikationsschemata ersetzt werden, wie beispielsweise die Generierung von entsprechenden eindeutigen Hashes.

Die Menge C_g besteht aus Quadrupeln, die wiederum jeweils eine Änderung zwischen zwei Revisionen in der Revisionshistorie beschreiben. Die Vorgängerrevision, auf der die Änderungen aufbauen, wird mittels r_x und die aus der Änderung resultierende neue Revision mittels r_y bezeichnet. Es ist per Definition ausgeschlossen, dass r_x und r_y gleich sind und somit auf die gleiche Revision verweisen. Damit aber auch bei der initialen Erstellung des Revisionsgraphen der erste Commit auf die gleiche Art und

Weise abgebildet werden kann wie alle folgenden, wird eine Null-Revision eingeführt. Diese kann nicht abgefragt werden und wird ausschließlich beim initialen Commit verwendet. Neben der Vorgänger- und der Nachfolgerrevision wird das Delta zwischen diesen beiden Revisionen durch C^+ und C^- beschrieben. Bei C^+ handelt es sich um die Menge der hinzugefügten Elemente zum Inhalt von r_x, um r_y zu erreichen. C^- beschreibt dementsprechend die Menge der Elemente, die gelöscht werden müssen, um vom Inhalt von r_x zum Inhalt von r_y zu gelangen. Die Schnittmenge zwischen C^+ und C^- muss immer die leere Menge ergeben, um eine Überschneidungsfreiheit zu gewährleisten. Im Gegensatz dazu muss die Vereinigung der beiden Mengen immer ungleich der leeren Menge sein. So ist jede Änderung durch mindestens eine Hinzufügung oder mindestens eine Löschung gekennzeichnet.

Zur Beschreibung von unterschiedlichen Entwicklungszweigen (Branches) innerhalb des Revisionsgraphen wird die Menge der verfügbaren Entwicklungszweige B_g wiederum als eine Menge an Quadrupeln beschrieben. Die zu einem Entwicklungszweig zugehörigen Revisionen werden mit der Menge R_b beschrieben. Die Revision eines Entwicklungszweiges, die selbst keinen Nachfolger besitzt, wird als Blatt des Entwicklungszweigs bezeichnet und mittels r_l identifiziert. Die Revision r_l muss daher auch immer in der Menge R_b enthalten sein. Der vollständige Inhalt des Blattes des Entwicklungszweigs wird in Υ_l vorgehalten. Auf dessen Basis kann im Folgenden dann eine Rekonstruktion von Revisionen vorgenommen werden, zu denen nur die Deltainformation bekannt ist. Die eindeutige Identifizierung des Entwicklungszweigs innerhalb des Revisionsgraphen wird in diesem vereinfachten Fall wiederum mittels einer natürlichen Zahl aus $\mathbb{N}^+$ vorgenommen.

Das Symbol T_g beschreibt die Menge der Tags im Revisionsgraphen. Die Beschreibung erfolgt ähnlich zu B_g, jedoch ohne die Menge an zugeordneten Revisionen. Jedes Tag besteht daher aus einem Tripel, das wiederum aus der getaggten Revision r_t, dem vollständigen Revisionsinhalt Υ_t von r_t und einem eindeutigen Identifikator n_t besteht.

Die eindeutige Identifizierbarkeit eines Revisionsgraphen wird mittels n_g sichergestellt. Hierbei handelt es sich wiederum aus Gründen der Vereinfachung um eine natürliche Zahl aus $\mathbb{N}^+$.

4.4.1.2 Vorgänger-/Nachfolgerbeziehungen

Die Berechnung von Vorgängern beziehungsweise Nachfolgern von Revisionen im Revisionsgraphen kann auf Basis der Menge der angewendeten Änderungen C_g durchgeführt werden. In Gleichung 4.8 sind die zugehörigen Funktionen für Vorgänger ($\boldsymbol{pred}_{\boldsymbol{\mathcal{G}}}(r_y)$) und Nachfolger ($\boldsymbol{succ}_{\boldsymbol{\mathcal{G}}}(r_x)$) definiert. Beide Funktionen geben als Ergebnis eine Menge von Revisionen aus der Menge der verfügbaren Revisionen R_g zurück. Dies ist notwendig, da eine Revision auf der einen Seite aufgrund von vorangegangener Zusammenführung von divergierten Entwicklungszweigen mehrere Vorgänger haben kann, auf der anderen Seite aber auch mehrere Nachfolger existieren können, da auf Basis der vorliegenden Revision ein neuer Entwicklungszweig erzeugt wurde.

$$\begin{aligned} \boldsymbol{pred}_{\mathcal{G}}(r_y) &:= \left\{ r_x \in R_g \;\middle|\; \left(r_x, r_y, C^+, C^-\right) \in C_g \right\} \\ \boldsymbol{succ}_{\mathcal{G}}(r_x) &:= \left\{ r_y \in R_g \;\middle|\; \left(r_x, r_y, C^+, C^-\right) \in C_g \right\} \end{aligned} \tag{4.8}$$

4.4.1.3 Pfadgenerierung und Deltawiederherstellung

Der Zugriff auf die Inhalte einzelner Revisionen erfordert Möglichkeiten, um aus den gespeicherten Deltas zwischen den Revisionen diesen Inhalt wiederherzustellen. Hierfür sind Mechanismen für die Generierung von Pfaden notwendig, um die Wiederherstellung durchführen zu können. Gleichung 4.9 stellt die zugehörigen mathematischen Beschreibungen dar. Es handelt sich aufgrund der Technologieunabhängigkeit an dieser Stelle vor allem um mögliche Interaktionen und die Definition von Rückgabewerten. Ein Pfad innerhalb eines Revisionsgraphen ist dabei stets als eine Sequenz von Änderungen als Subset von C_g beschrieben und kann mittels $r_x \rightarrow^* r_y$ dargestellt werden, wobei $\rightarrow^*$ angibt, dass eine beliebige Anzahl von Revisionen auf diesem Pfad liegen kann. Die zugehörige Funktion $\boldsymbol{path}_{\mathcal{G}}(r_x, r_y)$ gibt eine ungeordnete Liste an Elementen aus C_g zurück. Die Elemente definieren den Pfad zwischen der Startrevision r_x und der Zielrevision r_y. Für die Umsetzung können Algorithmen für die Identifizierung von kürzesten Wegen angewendet werden, wie beispielsweise von Dijkstra [Dij59] beschrieben. Der generierte Pfad kann anschließend verwendet werden, um die durchgeführten Änderungen nachzuverfolgen und den Inhalt einer Revision zurückzugeben, indem die Änderungen auf den letzten vollständig vorhandenen Inhalt kontinuierlich angewendet werden. Im Folgenden wird hierfür zur Abstraktion die Funktion $\boldsymbol{getContent}_{\mathcal{G}}(r_x)$ verwendet.

$$\begin{aligned} r_x \rightarrow^* r_y &= \boldsymbol{path}_{\mathcal{G}}(r_x, r_y) \subseteq C_g \\ \boldsymbol{path}_{\mathcal{G}}(r_x, r_y) &\ldots \textit{ Sequenz von Änderungen von } r_x \textit{ zu } r_y \\ \boldsymbol{getContent}_{\mathcal{G}}(r_x) &\ldots \textit{ Inhalt von Revision } r_x \end{aligned} \tag{4.9}$$

Die Wiederherstellung der Revisionsinhalte kann dabei in unterschiedlichen Richtungen im Revisionsgraphen vorgenommen werden. So kann beispielsweise von einem Blatt eines Entwicklungszweiges ausgehend eine vorangegangene Revision oder von der ersten Revision ausgehend eine nachfolgende Revision wiederhergestellt werden. Ebenso ist es denkbar, dass eine Kombination der beiden Richtungen auftritt, um eine Revision wiederherzustellen, wenn divergierte Entwicklungszweige oder Zusammenführungen von diesen stattgefunden haben. Damit immer eine eindeutige Wiederherstellung möglich ist, müssen die Änderungen zwischen zwei Revisionen nur das Delta zwischen diesen beschreiben und dürfen keine Elemente, die bereits existieren, noch einmal hinzufügen oder bereits gelöschte Elemente noch einmal löschen. Formalisiert ist diese Einschränkung in Gleichung 4.10 dargestellt. Sie gilt für alle Revisionen des Revisionsgraphen und für alle Vorgänger-Nachfolger-Beziehungen in diesem Revisionsgraphen. Υ_x bezeichnet hierbei den vollständigen Inhalt einer Revision r_x, die direkter Vorgänger der Revision r_y ist. Die Änderungen zwischen r_x und r_y werden mittels (r_x, r_y, C^+, C^-) beschrieben.

$$\begin{aligned} \Upsilon_x \cap C^+ &= \emptyset \\ \Upsilon_x \backslash C^- &= \emptyset \end{aligned} \tag{4.10}$$

Mit Hilfe der Funktion ***strip***, beschrieben in Gleichung 4.11, können die entsprechenden Elemente entfernt werden, bevor die Änderungen dem Revisionsgraphen hinzugefügt werden, wenn Änderungen vorliegen, die der Definition aus Gleichung 4.10 nicht entsprechen. Die Rückgabe ist ein Tupel, bestehend aus den Hinzufügungen und Löschungen, die der Definition aus Gleichung 4.10 genügen und damit dem Revisionsgraphen hinzugefügt werden können.

$$\begin{aligned} \boldsymbol{strip}\left(\Upsilon_x, C^+, C^-\right) &:= \left(C^+ \backslash \Upsilon_x, C^- \cap \Upsilon_x\right) \\ &= \left(C^+_{stripped}, C^-_{stripped}\right) \end{aligned} \tag{4.11}$$

4.4.1.4 Grundlegende Revisionskontrollfunktionalitäten

Für die Interaktion mit dem Revisionsgraphen werden weitere Funktionen benötigt, um diesen zum einen initial erstellen zu können, jedoch auch, um Änderungen an diesem durchzuführen. Die im Folgenden dargestellten Funktionen zur Interaktion mit dem Revisionsgraphen werden direkt auf den gegebenen Revisionsgraphen $\mathcal{G}$ angewendet und geben die modifizierte Instanz $\mathcal{G}$ bezeichnet mit $\mathcal{G}'$ zurück. Notwendige neue Bezeichner werden generiert, indem die Kardinalität der bereits bestehenden Menge um eins inkrementiert wird.

Initiale Erstellung und Löschung eines Revisionsgraphen Da innerhalb eines Revisionsverwaltungssystems mehrere Revisionsgraphen parallel existieren können, wird zusätzlich eine übergeordnete Menge Γ eingeführt, die alle vorhanden Revisionsgraphen vorhält. Die Erstellung eines neuen Revisionsgraphen erfolgt mittels der Funktion $\boldsymbol{create}_\Gamma$, wie in Gleichung 4.12 dargestellt. Für diesen initialen Commit können bereits mögliche Hinzufügungen spezifiziert werden. Das Ergebnis der Funktion ist ein neuer Revisionsgraph mit einer Revision, einer durchgeführten Änderung und einem neuen Entwicklungszweig. Die Änderung beschreibt die etwaigen Hinzufügungen, die bei der Erstellung angegeben werden. Der Entwicklungszweig stellt den *master*-Zweig dar, der die erstellte Revision beinhaltet.

$$\begin{aligned} \boldsymbol{create}_\Gamma\left(C^+\right) &:= \Gamma \cup \left\{\left(\{1\}, \left\{(0,1,C^+,\emptyset)\right\}, \left\{(\{1\},1,C^+,1)\right\}, \emptyset, |\Gamma|+1\right)\right\} \\ &= \Gamma \cup \{(R_g, C_g, B_g, T_g, n_g)\} = \Gamma \cup \{\mathcal{G}\} = \Gamma' \end{aligned} \tag{4.12}$$

Für den Fall, dass ein Revisionsgraph vollständig aus dem Revisionsverwaltungssystem Γ entfernt werden soll, steht die Funktion $\boldsymbol{drop}_\Gamma$ zur Verfügung. Gleichung 4.13 beschreibt die zugehörige Funktionalität mathematisch. Der durch n_g spezifizierte Revisionsgraph $\mathcal{G}$ wird dabei aus der Menge Γ entfernt.

$$\begin{aligned} \boldsymbol{drop}_{\Gamma}(n_g) &:= \Gamma \setminus \{\mathcal{G}\} \mid \mathcal{G} = (R_g, C_g, B_g, T_g, n_g) \\ &= \Gamma' \end{aligned} \tag{4.13}$$

Erstellung eines neuen Entwicklungszweiges Ein neuer Entwicklungszweig wird erstellt, indem die Funktion $\boldsymbol{branch}_{\mathcal{G}}$ angewendet wird. Die Vorgehensweise hierfür ist in Gleichung 4.14 beschrieben. Der neue Entwicklungszweig wird auf Basis der angegebenen Revision erzeugt. In Bezug auf den Revisionsgraphen wird nur die Menge B_g verändert. Dieser wird ein neues Quadrupel hinzugefügt, das aus der spezifizierten Revision r_x und dem vollständigen Inhalt Υ_x von r_x besteht, da diese Revision gleichzeitig auch das Blatt des Zweiges darstellt. Da es sich um das Anlegen eines neuen Zweiges handelt muss des Weiteren ein neuer Bezeichner generiert werden.

$$\begin{aligned} &\boldsymbol{branch}_{\mathcal{G}}(r_x) := \\ &\left(\begin{array}{c} R_g, \\ C_g, \\ B_g \cup \left\{ (\{r_x\}, r_x, \Upsilon_x, |B_g| + 1) \right\}, \\ T_g, \\ n_g \end{array} \right) \left| \begin{array}{l} \mathcal{G} = (R_g, C_g, B_g, T_g, n_g); \\ \Upsilon_x = \boldsymbol{getContent}_{\mathcal{G}}(r_x) \end{array} \right. \\ &\qquad = (R_g, C_g, B'_g, T_g, n_g) = \mathcal{G}' \end{aligned} \tag{4.14}$$

Erstellung eines neuen Tags Analog zur Erstellung von neuen Entwicklungszweigen können ebenfalls neue Tags mittels $\boldsymbol{tag}_{\mathcal{G}}$ erstellt werden. Gleichung 4.15 beschreibt die notwendigen Änderungen am Revisionsgraphen. Diese beschränken sich auf die Menge der Tags T_g, die um ein neues Tripel erweitert wird. Das Tripel enthält die spezifizierte Revision r_x, den vollständigen Inhalt Υ_x von r_x und einen neuen Bezeichner für den Tag.

$$
\begin{aligned}
&\boldsymbol{tag}_{\mathcal{G}}\left(r_x\right) := \\
&\left(\begin{array}{c} R_g, \\ C_g, \\ B_g, \\ T_g \cup \left\{(r_x, \Upsilon_x, |T_g|+1)\right\}, \\ n_g \end{array}\right) \left| \begin{array}{l} \mathcal{G} = (R_g, C_g, B_g, T_g, n_g); \\ \Upsilon_x = \boldsymbol{getContent}_{\mathcal{G}}\left(r_x\right) \end{array} \right. \\
&\qquad = \left(R_g, C_g, B_g, T'_g, n_g\right) = \mathcal{G}'
\end{aligned}
\tag{4.15}
$$

Erstellung eines neuen Commits Die Funktion $\boldsymbol{commit}_{\mathcal{G}}$ ist eine komplexe Operation, die Auswirkungen auf die Mengen R_g, C_g und B_g hat. Die durchzuführenden Änderungen (Hinzufügungen und Löschungen) werden in Bezug auf einen Entwicklungszweig und damit bezogen auf das Blatt von diesem angegeben. Im Weiteren wird zur Vereinfachung der Darstellung angenommen, dass diese Änderungen die Definition aus 4.10 erfüllen. Andernfalls muss vorher die Funktion aus Gleichung 4.11 angewendet werden. Für die Durchführung des Commits, wie in Gleichung 4.16 dargestellt, muss zuerst eine neue Revision r^* erstellt werden, die der Menge R_g hinzugefügt wird. Die Änderung mit den spezifizierten Hinzufügungen C^+ und Löschungen C^- zwischen dem Blatt r_l und der neuen Revision r^* wird der Menge C_g als neues Quadrupel hinzugefügt. Da r^* dem angegebenen Entwicklungszweig hinzugefügt werden muss, muss dieser entsprechend aktualisiert werden. Das wiederum erfordert die Löschung des bestehenden Entwicklungszweiges aus B_g und der anschließenden Hinzufügung des aktualisierten Quadrupels. Der Bezeichner n_b bleibt unverändert, da es sich um denselben Entwicklungszweig handelt. Die Revision r^* wird als neues Blatt hinzugefügt, was die Aktualisierung des vollständigen Inhalts Υ_l mit den spezifizierten Änderungen erfordert.

$$
\boldsymbol{commit}_{\mathcal{G}}\left(n_b,C^+,C^-\right) :=
\left(\begin{array}{c}
R_g \cup \{r^*\}, \\
C_g \cup \left\{(r_l,r^*,C^+,C^-)\right\}, \\
B_g \backslash \{b\} \cup \left\{(R_b \cup \{r^*\}, r^*, \\
(\Upsilon_l \cup C^+) \backslash C^-, n_b)\right\}, \\
T_g, \\
n_g
\end{array}\right|
\begin{array}{l}
\mathcal{G} = (R_g,C_g,B_g,T_g,n_g); \\
b \in B_g; \\
b = (R_b,r_l,\Upsilon_l,n_b); \\
C^+ \cap\ C^- = \emptyset; \\
C^+ \cup\ C^- \neq \emptyset; \\
r^* = |R_g| + 1
\end{array}
= (R'_g,C'_g,B'_g,T_g,n_g) = \mathcal{G}' \tag{4.16}
$$

Revidieren eines Commits Mit Hilfe der Funktion $\boldsymbol{revert}_{\mathcal{G}}$, wie in Gleichung 4.17 dargestellt, kann ein vorangegangener Commit rückgängig gemacht werden. Die Identifizierung erfolgt mittels dem zugehörigen Entwicklungszweig, da immer nur der letzte Commit des Entwicklungszweiges rückgängig gemacht werden kann. Da Revisionen zu mehreren Entwicklungszweigen gehören und mehrere Vorgänger haben können, wird die vorausgehende Revision r_p aus der Menge der möglichen Revisionen R_p herausgefiltert. Diese Filterung wird mittels einer Prüfung vorgenommen, die sicherstellt, dass die vorangehende Revision ebenfalls auf dem angegebenen Entwicklungszweig liegt. Beim Revidieren erfolgt keine Löschung des vorangegangenen Commits. Die zugehörigen Hinzufügungen und Löschungen werden jedoch rückgängig gemacht, indem in einem neuen Commit die Hinzufügungen gelöscht und die Löschungen hinzugefügt werden. Das erlaubt auf der einen Seite, dass alle Änderungen weiterhin nachvollziehbar sind und auf der anderen Seite kann diese Information durch Algorithmen zur Wiederherstellung von vollständigen Revisionsinhalten genutzt werden. Dadurch kann dieser Wiederherstellungsprozess optimiert werden, indem eine Aufeinanderfolge von $\boldsymbol{commit}_{\mathcal{G}}$ und $\boldsymbol{revert}_{\mathcal{G}}$ im Rekonstruktionsprozess übersprungen werden kann.

$$
\boldsymbol{revert}_{\mathcal{G}}\left(n_b\right) :=
$$

$$
\left(\begin{array}{c}
R_g \cup \{r^*\}, \\
C_g \cup \left\{\left(r_l, r^*, C^-, C^+\right)\right\}, \\
B_g \setminus \{b\} \cup \left\{\left(R_b \cup \{r^*\}, r^*, \right.\right. \\
\left.\left.\left(\Upsilon_l \cup C^-\right) \setminus C^+, n_b\right)\right\}, \\
T_g, \\
n_g
\end{array}\right) \left| \begin{array}{l}
\mathcal{G} = (R_g, C_g, B_g, T_g, n_g); \\
b \in B_g; \\
b = (R_b, r_l, \Upsilon_l, n_b); \\
\boldsymbol{pred}_{\mathcal{G}}\left(r_l\right) = R_p; \\
r_p \in R_p \cap R_b; \\
\left(r_p, r_l, C^+, C^-\right) \in C_g; \\
r^* = |R_g| + 1
\end{array}\right. \tag{4.17}
$$

$$
= \left(R'_g, C'_g, B'_g, T_g, n_g\right) = \mathcal{G}'
$$

4.4.1.5 Semantische Beschreibung

Die in den vorangegangenen Abschnitten eingeführten Revisionskontrollfunktionalitäten werden im Folgenden semantisch beschrieben (Anforderung A-204). Hierfür werden Konzepte aus der PROV Ontology (PROV-O) als Grundlage wiederverwendet. Das zugehörige UML-Modell ist in Abbildung 4.9 dargestellt. Zentrales Element bildet dabei wie bereits in der mathematischen Beschreibung der *RevisionGraph*, der einem *RevisionControlSystem* zugeordnet ist. Als zentrales Element hält der *RevisionGraph* alle *Commit*- und *Entity*-Elemente vor. Einem *Commit* kann Metainformation zugeordnet werden, wie beispielhaft mittels einer Nachricht, einem Zeitstempel und einem assoziierten Nutzer dargestellt. Darüber hinaus ist es ebenfalls möglich, diesem Commit ein Attribut für Abwärtskompatibilität zuzuordnen. Auf dieses wird innerhalb dieser Arbeit verzichtet, da nach Heflin und Pan [HP04] die Kennzeichnung, dass eine Änderung abwärtskompatibel ist, von zusätzlichem Wissen abhängt. Die einzelnen Spezialisierungen des Commits ermöglichen die Beschreibung der durchgeführten Aktionen, die wiederum den in Abschnitt 4.4.1.4 eingeführten grundlegenden Funktionalitäten entsprechen. Jeder dieser Spezialisierungen besitzt dabei unterschiedliche assoziierte Elemente, die genutzt werden, um die durchgeführte Änderung zu beschreiben. Ein *used* gibt dabei stets an, welche Revision als Grundlage für die Aktion dient und mittels *generated* wird das korrespondierende Resultat beschrieben. Entitäten im Revisionsgraphen unterteilen sich in *Reference* und *Revision*. Referenzen beschreiben dabei sowohl Tags als auch Entwicklungszweige und sind über einen Identifier eindeutig identifizierbar. Jeder Referenz ist außerdem ein vollständiger Inhalt zugeordnet, der in diesem Fall als eine nicht näher spezifizierte Menge an Elementen realisiert ist, um die Technologieunabhängigkeit zu gewährleisten. Revisionen sind im Revisionsgraphen über einen eindeutigen Identifier referenzierbar.

Die Historie der Revisionen und dementsprechend auch die Vor- und Nachfolgebeziehung wird mittels der Assoziationen *wasDerivedFrom* angegeben. Weiterhin wird den Commits, die Revisionen erzeugen, ein *ChangeSet* zugeordnet, das das Delta in Bezug auf die vorangehende Revision beschreibt. Dieses setzt sich aus einer Referenz zu einem Add- und einem Delete-Set zusammen, wobei es sich jeweils wiederum um eine nicht näher spezifizierte Menge an Elementen (*Statement*) handelt. Das *ChangeSet* referenziert außerdem die vorangegangene Revision und die erzeugte Revision, um die Historie zu beschreiben.

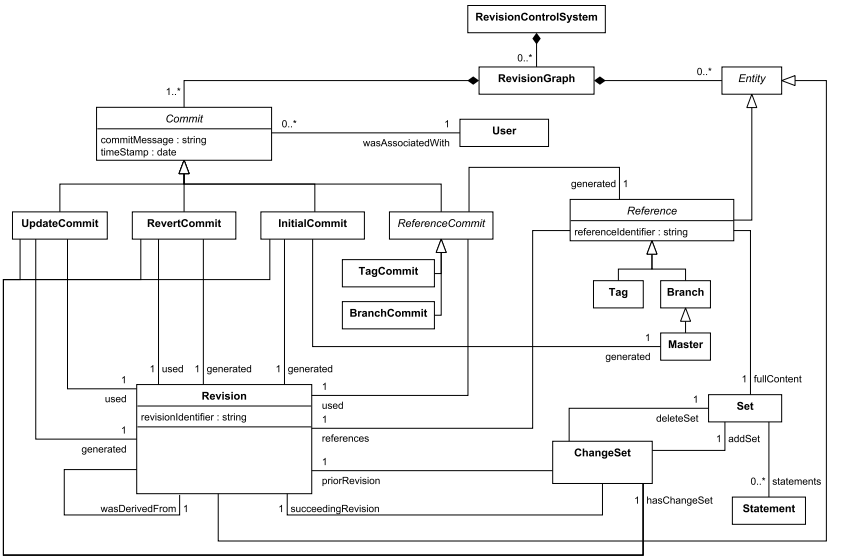


Abbildung 4.9: Semantische Beschreibung der Revisionskontrollfunktionalitäten als UML-Modell

4.4.2 Aggregation von High-Level-Changes

Mechanismen für die Aggregation von atomaren Änderungen zu High-Level-Changes (Anforderung A-203) werden unter anderem in [Keh15] und [Pap+13] beschrieben. Im Folgenden wird von diesen technologiespezifischen Konzepten abstrahiert, um auch für die weiteren Arbeiten eine technologieunabhängige Grundlage zu schaffen. Hierfür werden wiederum mathematische und semantische Beschreibungen entwickelt, die sich in die für die Revisionskontrolle bereits beschriebenen eingliedern. In einer späteren Umsetzung können dann die bestehenden Möglichkeiten, wie [Keh15] und [Pap+13], für die Aggregation genutzt werden.

4.4.2.1 Mathematische Beschreibung

Die Aggregation von atomaren Änderungen zu High-Level-Changes erfolgt mittels der Funktion $\boldsymbol{hlcAgg_{\mathcal{G}}}$, wie in Gleichung 4.18 dargestellt. Ausgangspunkt für die Berechnung der zugehörigen High-Level-Changes sind die Änderungen zwischen zwei Revisionen r_x und r_y. Auf Basis von diesen gibt die Funktion $\boldsymbol{\Phi_{\mathcal{G}}}$, beschrieben in Gleichung 4.19, die High-Level-Changes zurück. Jede dieser Aggregationen besteht aus einem Tripel, das die verwendeten hinzugefügten und gelöschten Elemente, sowie einen eindeutigen Identifikator n_z beschreibt. Mittels des Kennzeichnens n_z kann der erkannten semantischen Änderung des Weiteren die entsprechende Bedeutung zugeordnet werden. Dies kann beispielsweise mittels eines Verweises auf den zugrunde liegenden Regelsatz erfolgen. Hierbei stellt bereits Papavasileiou [Pap+13] nach Klein [Kle04] fest, dass es nicht möglich ist, eine vollständige Liste an Regeln zu spezifizieren, da es keine allgemeingültige Menge an Änderungsoperationen gibt, auf denen diese basieren könnte. Die technologiespezifische Umsetzung von $\boldsymbol{\Phi_{\mathcal{G}}}$ kann dementsprechend unterschiedliche Ausprägungen besitzen. Je nach Güte des Regelsatzes ist es möglich, dass alle atomaren Änderungen zu High-Level-Changes aggregiert werden können oder nach Anwendung der Regeln Elemente bestehen bleiben, die nicht zuzuordnen sind. Diese verbleibenden Elemente werden in $\boldsymbol{hlcAgg_{\mathcal{G}}}$ mittels C_r^+ und C_r^- beschrieben.

$$
\begin{aligned}
&\boldsymbol{hlcAgg_{\mathcal{G}}}\left(r_x,r_y\right) := \\
&\left(\begin{array}{c} \boldsymbol{\Phi_{\mathcal{G}}}\left(r_x,r_y\right), \\ C^+ \backslash \left(\bigcup C^+_{sub_{n_z}}\right), \\ C^- \backslash \left(\bigcup C^-_{sub_{n_z}}\right) \end{array}\right| \begin{array}{l} \mathcal{G} = (R_g,C_g,B_g,T_g,n_g); \\ \forall \left(C^+_{sub_{n_z}},C^-_{sub_{n_z}},n_z\right) \in \boldsymbol{\Phi_{\mathcal{G}}}\left(r_x,r_y\right); \\ \left(r_x,r_y,C^+,C^-\right) \in C_g \end{array} \\
&\qquad = \left(\boldsymbol{\Phi_{\mathcal{G}}}\left(r_x,r_y\right),C_r^+,C_r^-\right)
\end{aligned} \tag{4.18}
$$

$$
\boldsymbol{\Phi_{\mathcal{G}}}\left(r_x,r_y\right) = \left\{ \left(C^+_{sub_{n_z}},C^-_{sub_{n_z}},n_z\right) \middle| \begin{array}{l} \mathcal{G} = (R_g,C_g,B_g,T_g,n_g); \\ \left(r_x,r_y,C^+,C^-\right) \in C_g; \\ C^+_{sub_{n_z}} \subseteq C^+;\ C^-_{sub_{n_z}} \subseteq C^-; \\ n_z \in \mathbb{N}^+ \end{array} \right\} \tag{4.19}
$$

4.4.2.2 Semantische Beschreibung

Das in Abbildung 4.10 dargestellte UML-Modell erweitert die in Abbildung 4.9 eingeführte Klasse *ChangeSet* um die High-Level-Change-Aggregation und stellt damit die semantische Beschreibung der mathematischen Grundlagen aus Abschnitt 4.4.2.1

sicher (Anforderung A-204). Es handelt sich dabei um eine Integrationsschnittstelle zur Integration von Aggregationsmechanismen, wie in [Pap+13] oder [Keh15] präsentiert. Jedem *ChangeSet* können beliebig viele *SemanticChange*-Elemente zugeordnet werden, die wiederum aus einer Menge an *additions* und einer Menge an *deletions* besteht. Hierdurch können die zugehörigen atomaren Änderungen beschrieben werden, die durch den beschriebenen *SemanticChange* aggregiert werden. Die Menge der atomaren Änderungen, die nicht einem *SemanticChange* zugeordnet werden können, wird nicht beschrieben, da die atomaren Änderungen vollständig als Menge vorliegen und daher über Differenzbildung der nicht zuzuordnende Rest berechnet werden kann. *SemanticChange*s, die sich gegenseitig aufheben, können mittels der Relation *inverts* beschrieben werden. Die semantische Beschreibung der einzelnen zu integrierenden Aggregationsmechanismen erfolgt als Ableitung von *SemanticChange*, wodurch unterhalb von dieser Klasse die verschiedenen Ausprägungen beschrieben werden können. Des Weiteren ist es auf der resultierenden Ebene möglich, auf entsprechende Regelsätze zu verweisen, um die Verknüpfung zwischen angewendeter Regel und entdecktem High-Level-Change herzustellen.

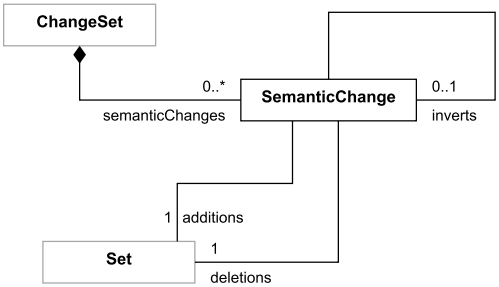


Abbildung 4.10: Erweiterung der semantischen Beschreibung um die Aggregation von High-Level-Changes als UML-Modell

4.4.3 Zusammenführung divergierter Entwicklungszweige

Die Zusammenführung von divergierten Entwicklungszweigen (Anforderung A-202) spielt insbesondere in kollaborativen Umgebungen mit parallelen Entwicklungen eine wichtige Rolle, um beispielsweise die Ergebnisse vor einer Veröffentlichung wieder zusammenführen zu können. In den folgenden Abschnitten werden unterschiedliche Methoden der Zusammenführung aber auch Lösungen für die Konflikterkennung und -behebung bei der Zusammenführung beschrieben.

4.4.3.1 Methoden der Zusammenführung

Für die Zusammenführung von divergierten Entwicklungszweigen stehen in Systemen wie git unterschiedliche Möglichkeiten zur Verfügung. Im Folgenden werden an git und [AM17]

angelehnte Möglichkeiten zur Zusammenführung technologieunabhängig beschrieben. Basis bilden dabei die in Abschnitt 4.4.1 eingeführten mathematischen Beschreibungen des Revisionsgraphen.

3-Wege-Merges Ein weit verbreitetes Szenario bei der Zusammenführung sind sogenannte 3-Wege-Merges. Die zugehörige Funktion $\boldsymbol{merge}_{\mathcal{G}}$ ist in Gleichung 4.20 beschrieben. Dabei wird der Quellentwicklungszweig (b_s) in den Zielentwicklungszweig (b_t) zusammengeführt. Das Ergebnis ist eine neue Revision mit zwei Vorgängern, wobei diese Revision den zusammengeführten Inhalt der beiden Entwicklungszweige darstellt, in dem alle Konflikte behoben sind. Der Revisionsgraph $\mathcal{G}$ wird mittels der neuen Revision r^* aktualisiert. Für beide betroffenen Entwicklungszweige müssen die notwendigen Änderungen zwischen dem Blatt des Zweiges r_{l_s} beziehungsweise r_{l_t} und r^* beschrieben werden. Hierfür wird jeweils ein neues Quadrupel in C_g angelegt. Der Inhalt der Änderungen ist abhängig von der gewählten Strategie der Zusammenführung und wird an dieser Stelle mittels der Funktion $\boldsymbol{\Psi_G}$, beschrieben in Gleichung 4.21, abstrahiert. Die Funktion gibt dabei die notwendigen Änderungen für beide Zweige als Tupel zurück, wobei diese jeweils der Definition aus 4.10 genügen. Für die Realisierung stehen unterschiedliche Ansätze, wie beispielsweise in [Fai+16] vorgestellt, zur Verfügung. Auf Möglichkeiten zur Umsetzung wird in Abschnitt 4.4.3.2 eingegangen.

$$
\begin{aligned}
&\boldsymbol{merge}_{\mathcal{G}}\left(n_{b_s},n_{b_t}\right) := \\
&\left(\begin{array}{c}
R_g \cup \{r^*\}, \\
C_g \cup \left\{(r_{l_s},r^*,C_s^+,C_s^-),(r_{l_t},r^*,C_t^+,C_t^-)\right\}, \\
B_g \setminus \{b_t\} \cup \left\{\left(R_{b_t} \cup \{r^*\}, r^*, \right.\right. \\
\left.\left.(\Upsilon_{l_t} \cup C_t^+) \setminus C_t^-, n_{b_t}\right)\right\}, \\
T_g, \\
n_g
\end{array}\right) \left|\begin{array}{l}
\mathcal{G} = (R_g,C_g,B_g,T_g,n_g); \\
b_s \in B_g;\ b_t \in B_g; \\
b_s = (R_{b_s},r_{l_s},\Upsilon_{l_s},n_{b_s}); \\
b_t = (R_{b_t},r_{l_t},\Upsilon_{l_t},n_{b_t}); \\
r^* = |R_g| + 1; \\
\boldsymbol{\Psi}_{\mathcal{G}}\left(n_{b_s},n_{b_t}\right)
\end{array}\right. \\
&\quad = (R'_g,C'_g,B'_g,T_g,n_g) = \mathcal{G}'
\end{aligned}
\tag{4.20}
$$

$$
\boldsymbol{\Psi}_{\mathcal{G}}\left(n_{b_s},n_{b_t}\right) = \left((C_s^+,C_s^-),(C_t^+,C_t^-)\right) \tag{4.21}
$$

Pick Bereits durchgeführte Änderungen können mittels der Funktion $\boldsymbol{pick}_{\mathcal{G}}$, beschrieben in Gleichung 4.22, wiederverwendet werden. Hierbei werden die bereits durchgeführten Änderungen, die zu einer Revision r_p geführt haben, genutzt und auf das Blatt eines Entwicklungszweiges b angewendet. Durch die Anwendung der Funktion wird eine neue Revision r^* mit den zugehörigen Änderungen zwischen r_l und r^* erzeugt. Es handelt sich

dabei um eine Kopie der Hinzufügungen und Löschungen, die zur Revision r_p geführt haben. Zur Erfüllung der Definition aus Gleichung 4.10 muss jedoch vorher ein ***strip*** in Bezug auf den vollständigen Inhalt des Blatts Υ_l durchgeführt werden. Des Weiteren muss der Entwicklungszweig mit dem neuen Blatt aktualisiert werden. Ein Pick kann nur durchgeführt werden, wenn die Kardinalität des Vorgängers von r_p gleich eins ist. Zusammengeführte Revisionen können dementsprechend nicht für einen Pick genutzt werden, da eine zusätzliche Auswahl getroffen werden muss, welche der beiden möglichen Änderungen genutzt werden sollen.

$$
\begin{aligned}
&\boldsymbol{pick}_{\mathcal{G}}\left(r_p,n_b\right) := \\
&\left(\begin{array}{c}
R_g \cup \{r^*\}, \\
C_g \cup \left\{(r_l,r^*,C_s^+,C_s^-)\right\}, \\
B_g \backslash \{b\} \cup \left\{(R_b \cup \{r^*\}, r^*, \right. \\
\left. (\Upsilon_l \cup C_s^+) \backslash C_s^-, n_b)\right\}, \\
T_g, \\
n_g
\end{array}\right| \begin{array}{l}
\mathcal{G} = (R_g,C_g,B_g,T_g,n_g); \\
r_p \in R_g;\ |\boldsymbol{pred}_{\mathcal{G}}\left(r_p\right)| = 1; \\
(\boldsymbol{pred}_{\mathcal{G}}\left(r_p\right), r_p, C^+, C^-) \in C_g; \\
C_s^+ = C^+ \backslash \Upsilon_l; \\
C_s^- = C^- \cap \Upsilon_l; \\
b \in B_g;\ b = (R_b,r_l,\Upsilon_l,n_b); \\
r^* = |R_g| + 1
\end{array} \\
&\quad = \ (R'_g,C'_g,B'_g,T_g,n_g) = \mathcal{G}'
\end{aligned}
\tag{4.22}
$$

Das einfache Pick (siehe Gleichung 4.22) kann zu einem Pick erweitert werden, das es erlaubt, mehrere Revisionen zu spezifizieren, wie in Gleichung 4.23 dargestellt. Als Eingabegrößen sind hierbei eine geordnete Liste von Revisionen und der Zielentwicklungszweig notwendig. Anschließend kann der einfache Pick aus Gleichung 4.22 in der durch die Liste angegebenen Reihenfolge für die einzelnen Revisionen angewendet werden, wie es in Gleichung 4.23 in Pseudocode angegeben ist.

$$
\begin{aligned}
&\mathcal{G} = (R_g,C_g,B_g,T_g,n_g);\ \vec{R_p} := [r_{p1},r_{p2},\ldots,r_{pm}];\ m \in \mathbb{N}^+; \\
&b \in B_g;\ b = (R_b,r_l,\Upsilon_l,n_b) \\
&\boldsymbol{pick}_{\mathcal{G}}\left(\vec{R_p},n_b\right): \\
&\quad \boxed{\begin{array}{ll}
1: & \text{FOR } r_p \text{ IN } \vec{R_p}: \\
2: & \quad \boldsymbol{pick}_{\mathcal{G}}\left(r_p,n_b\right);
\end{array}}
\end{aligned}
\tag{4.23}
$$

Fast Forward Die Funktion $\boldsymbol{fastForward_{\mathcal{G}}}$ ist eine weitere Möglichkeit der Zusammenführung von divergierten Entwicklungszweigen. Sie wird eingesetzt, um die Revisionshistorie zu glätten und nicht notwendige 3-Wege-Merges zu vermeiden. Ein Fast Forward ändert dafür den Zeiger eines Entwicklungszweiges auf den Zeiger eines anderen Entwicklungszweiges. Hierfür muss das Blatt des ersten Zweiges als Revision auch im zweiten Entwicklungszweig enthalten sein. Gleichung 4.24 gibt hierzu die formale Beschreibung der durchzuführenden Änderungen am Revisionsgraphen an. Die Mengen R_g, C_g and T_g bleiben dabei unverändert, da nur der Zeiger angepasst wird. Der Zeiger des Entwicklungszweigs b_t wird dabei auf den Zeiger von b_s vorgeschoben. Hierfür müssen alle Revisionen auf dem Pfad vom Blatt r_{l_t} zu r_{l_s} als zugehörige Revisionen zu b_t hinzugefügt, sowie das Blatt und der zugehörige vollständige Inhalt angepasst werden.

$$
\begin{aligned}
&\boldsymbol{fastForward_{\mathcal{G}}}\,(n_{b_s},n_{b_t}) := \\
&\left(\begin{array}{c} R_g, \\ C_g, \\ B_g \setminus \{b_t\} \cup \left\{(R_{b_t} \cup P, r_{l_s}, \Upsilon_{l_s}, n_{b_t})\right\}, \\ T_g, \\ n_g \end{array}\right) \left| \begin{array}{l} \mathcal{G} = (R_g,C_g,B_g,T_g,n_g); \\ b_s \in B_g;\ b_t \in B_g; \\ b_s = (R_{b_s},r_{l_s},\Upsilon_{l_s},n_{b_s}); \\ b_t = (R_{b_t},r_{l_t},\Upsilon_{l_t},n_{b_t}); \\ r_{l_t} \in R_{b_s}; \\ P := \{r_y \,|(r_x,r_y,C^+,C^-) \\ \qquad\qquad \in \boldsymbol{path_G}\,(r_{l_t},r_{l_s})\} \end{array}\right. \\
&\qquad = \ (R_g,C_g,B'_g,T_g,n_g) = \mathcal{G}'
\end{aligned}
\tag{4.24}
$$

4.4.3.2 Konflikterkennung und -behebung

Bei der Zusammenführung von divergierten Zweigen können Konflikte auftreten, die entsprechend erkannt und behoben werden müssen. Technologiespezifische Lösungen wurden bereits in vorangegangenen Arbeiten wie [Fai+16; HGU16; Keh15; Pap+13] entwickelt. Auf Basis dieser Ansätze erfolgt an dieser Stelle die Beschreibung einer technologieunabhängigen Möglichkeit für die Umsetzung der Gleichung 4.21 zur Spezifikation der Zusammenführung von zwei Entwicklungszweigen. Wie in [Keh15] und [HGU16] gefordert, werden dabei auch transiente Effekte in der Revisionshistorie beachtet. Bei diesen transienten Effekten handelt es sich um Lösch- und Hinzufügeoperationen in der Revisionshistorie, die vorangegangene Änderungen wiederum rückgängig machen [Keh15]. Die hierfür notwendigen mathematischen Grundlagen basieren auf [HGU16] und werden im Folgenden in das mathematische Gesamtgefüge dieser Arbeit eingegliedert.

Nachvollziehung von transienten Effekten auf atomarer Änderungsebene Beim Vergleich der Revisionshistorien von zwei Entwicklungszweigen können die einzelnen Elemente auf Ebene der atomaren Änderungen die in Gleichung 4.25 dargestellten Status aufweisen. Dieser Status kann entweder *gelöscht* (−), *hinzugefügt* (+) oder *keine Änderung erfahren* (0) sein. In $\mathcal{K}$ ist des Weiteren der Status *nicht enthalten* ($\emptyset$) aufgeführt, dieser kommt jedoch erst beim Vergleich von zwei Entwicklungszweigen zur Anwendung und ist an dieser Stelle nur zur vereinfachten mathematischen Definition mit aufgeführt.

$$\begin{aligned}\mathcal{K} := \{ & - = \text{“Gelöscht”},\\ & 0 = \text{“Keine Änderung erfahren”},\\ & + = \text{“Hinzugefügt”}\\ & \emptyset = \text{“Nicht enthalten”}\}\end{aligned} \tag{4.25}$$

Die Parameter von $\boldsymbol{\Psi}_{\mathcal{G}}\,(n_{b_s},n_{b_t})$ stellen die beiden Identifikatoren der zusammenzuführenden Entwicklungszweige dar. Somit kann im Folgenden davon ausgegangen werden, dass $b_s = (R_{b_s},r_{l_s},\Upsilon_{l_s},n_{b_s})$ und $b_t = (R_{b_t},r_{l_t},\Upsilon_{l_t},n_{b_t})$ gilt. Um die Änderungen zwischen den beiden Entwicklungszweigen vergleichen zu können, muss im ersten Schritt eine gemeinsame Vorgängerrevision gefunden werden. Zur Vereinfachung wird angenommen, dass die in Gleichung 4.26 dargestellte Funktion die Revision $r_c \in R_g$ zurückgibt.

$$\boldsymbol{getCommonAncestor}_{\mathcal{G}}\,(n_{b_s},n_{b_t}) = r_c \tag{4.26}$$

Mit dem berechneten r_c kann im Anschluss die in Gleichung 4.9 beschriebene Funktion zur Berechnung des Pfades zwischen zwei Revisionen wiederverwendet werden. Entsprechend wird diese Funktion für beide Entwicklungszweige angewendet, um die jeweilige Sequenz an Änderungen zu erhalten. Das Ergebnis sind die beiden in Gleichung 4.27 dargestellten Mengen C_{path_s} und C_{path_t}.

$$\begin{aligned}\boldsymbol{path}_{\mathcal{G}}\,(r_c,r_{l_s}) &= C_{path_s} \subset C_g\\ \boldsymbol{path}_{\mathcal{G}}\,(r_c,r_{l_t}) &= C_{path_t} \subset C_g\end{aligned} \tag{4.27}$$

Zu Beginn des anschließenden Nachvollziehens der Änderungen muss auf Basis des Inhaltes der gemeinsamen Revision r_c eine Startmenge Ω_{Start} für jeden Zweig gebildet werden. Im Anschluss erfolgt auf diese Menge die Anwendung der einzelnen Änderungen über den gebildeten Pfad hinweg. Die Startmenge setzt sich aus dem Inhalt von enthaltenen Statements und dem initialen Zustand *keine Änderung erfahren* zusammen. Dieser Zustand ist unabhängig von der vorangehenden Revisionsgeschichte, da r_c den Ausgangspunkt bildet. In Gleichung 4.28 ist die zugehörige mathematische Beschreibung angegeben.

$$\Omega_{Start_s} = \Omega_{Start_t} = \{(s,0) \in (\mathcal{S} \times \mathcal{K}) | s \in \boldsymbol{getContent}_{\mathcal{G}}\,(r_c)\} \tag{4.28}$$

Für die Aktualisierung der jeweiligen Zustände stehen die in Gleichung 4.29 dargestellten Funktionen $\boldsymbol{add}(\Omega,s)$ und $\boldsymbol{del}(\Omega,s)$ zur Verfügung. Durch diese kann der Zustand eines Statements aktualisiert werden. Basis bildet das vorangehende Ω auf dem Pfad beziehungsweise der Revision. In Gleichung 4.30 ist hierfür das allgemeine Verfahren für die Anwendung von allen Änderungen einer Revision r_{y_n} mittels der Funktion F dargestellt. Im Folgenden gilt, dass $\mathbb{N}_0^+$ der Menge der natürlichen Zahlen größer gleich Null entspricht. Die Änderungen werden durch ein entsprechendes Element $C_n = (r_{x_n},r_{y_n},C_n^+,C_n^-)$ aus dem erstellten Pfad vorgenommen, wobei $n \in \mathbb{N}_0^+$ und im Bereich $0 \leq n \leq |C_{path_s}| - 1$ beziehungsweise $0 \leq n \leq |C_{path_t}| - 1$ liegt. Falls das C_n^+ beziehungsweise C_n^- gleich der leeren Menge ist, so wird die Anwendung von $\boldsymbol{add}(\Omega,s)$ beziehungsweise $\boldsymbol{del}(\Omega,s)$ übersprungen. Die Anzahl der Funktionsaufrufe ergibt sich aus der Anzahl der Elemente in C_n^+ und C_n^-. Hierfür gilt für C_n^+, dass $\bar{a} = |C_n^+|$, $s_a^+ \in C_n^+$ im Bereich $0 \leq a \leq \bar{a} - 1$, wobei $a \in \mathbb{N}_0^+$. Analog gilt für C_n^-, dass $\bar{d} = |C_n^-|$, $s_d^- \in C_n^-$ im Bereich $0 \leq d \leq \bar{d} - 1$, wobei $d \in \mathbb{N}_0^+$. Die Anwendung entlang des Pfades ist in Gleichung 4.31 beschrieben. Die einzelnen C_n werden dabei entlang des Pfades, gestartet bei Revision r_c, geordnet verwendet, um die Historie entsprechend abbilden zu können. Dieses Vorgehen muss für beide Entwicklungszweige mit C_{path_s} und C_{path_t} angewendet werden.

$$\begin{aligned} \boldsymbol{add}(\Omega,s) &: \mathcal{P}(\mathcal{S} \times \mathcal{K}) \times \mathcal{S} \to \mathcal{P}(\mathcal{S} \times \mathcal{K}) : \\ &\Omega \mapsto (\Omega \backslash \{(s,-),(s,0)\}) \cup \{(s,+)\} \\ \boldsymbol{del}(\Omega,s) &: \mathcal{P}(\mathcal{S} \times \mathcal{K}) \times \mathcal{S} \to \mathcal{P}(\mathcal{S} \times \mathcal{K}) : \\ &\Omega \mapsto (\Omega \backslash \{(s,+),(s,0)\}) \cup \{(s,-)\} \end{aligned} \tag{4.29}$$

$$\boldsymbol{F_{C_n}}(\Omega) = \boldsymbol{add}(...(\boldsymbol{add}(\boldsymbol{del}(...(\boldsymbol{del}(\Omega,s_0^-),...),s_{\bar{d}-1}^-),s_0^+),...),s_{\bar{a}-1}^+) \tag{4.30}$$

$$\Omega_{End} = (\boldsymbol{F_{C_{n-1}}} \circ \boldsymbol{F_{C_{n-2}}} \circ ... \circ \boldsymbol{F_{C_0}})(\Omega_{Start}) \tag{4.31}$$

Erkennung struktureller Konflikte Auf Basis, der im vorangegangenen Abschnitt beschriebenen Nachvollziehung der transienten Effekte können bereits Konflikte erkannt werden. Hierfür müssen die beiden resultierenden Ω_{End} von Quell- und Zielentwicklungszweig miteinander verglichen werden. Je nach Anwendungsfall ist hierbei zu definieren, welche gegenläufigen Änderungen einen Konflikt hervorrufen, wie beispielsweise in [HGU16] dargestellt. Hieraus kann eine allgemeine Konfliktbeschreibungsmatrix abgeleitet werden, die in Abbildung 4.11 dargestellt ist.

Die einzelnen Status von Quell- und Zielentwicklungszweig beziehen sich auf die möglichen Status aus Gleichung 4.25. Durch die gegebenen Definitionen für die Erstellung von Ω_{End} und die allgemeinen Definitionen zur Beschreibung der Revisionskontrolle ist es nicht möglich, dass die Kombination bestehend aus $\emptyset$ und 0 auftritt. Dies gilt, da immer von einer gemeinsamen Revision der beiden zu vergleichenden Entwicklungszweige ausgegangen wird und damit ein Element entweder in dieser Revision enthalten ist oder

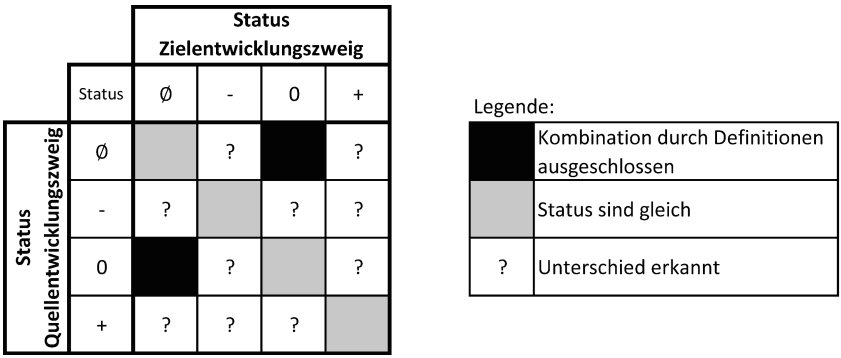


Abbildung 4.11: Allgemeine Konfliktbeschreibungsmatrix (in Anlehnung an [HGU16, S. 7])

nicht. Bei der Nachvollziehung über die weitere Historie hinweg kann der Status nur durch die Funktionen aus Gleichung 4.29 verändert werden. Dabei sind nur Aktualisierungen auf die Status − und + möglich. Die Kombinationen, bei denen der jeweilige Status eines Elementes gleich ist, geben an, dass bei diesen Elementen kein Unterschied vorliegt. Insbesondere wenn die Status gleich ∅ beziehungsweise 0 sind, haben die Elemente keinerlei Änderungen in der betrachteten Historie erfahren. Alle weiteren Kombinationen, die in Abbildung 4.11 mit einem Fragezeichen gekennzeichnet sind, weisen hingegen auf einen Unterschied zwischen Quell- und Zielentwicklungszweig hin. An dieser Stelle kann je nach Anwendungsfall entschieden werden, was davon einen Konflikt darstellt. Unter Umständen kann auch auf dieser Ebene definiert werden, wie der entsprechende Konflikt behoben werden kann. Die folgende Gleichung 4.32 ermöglicht die Definition dieser Regeln als ein Quadrupel. Mittels k_s und k_t können die Status für Quell- und Zielentwicklungszweig angegeben werden. Die Angabe q ermöglicht die Spezifikation, ob es sich um einen Konflikt handelt ($q = 1$), der eventuell manuell nachbearbeitet werden muss, oder um einen Unterschied ($q = 0$), der automatisch behoben werden kann. Für eine automatische Behebung wird des Weiteren der resultierende Status benötigt, der durch k_r beschrieben wird. Es handelt sich dabei um ein Subset von $\mathcal{K}$, da für die zusammengeführte Revision beschrieben werden muss, ob das Element enthalten sein soll oder nicht. Dafür eignen sich die Status + beziehungsweise − aus $\mathcal{K}$.

$$\mathcal{Q} = \{(k_s, k_t, q, k_r) | k_s, k_t \in \mathcal{K}; q \in \{0{,}1\}; k_r \in \{-,+\} \subset \mathcal{K}\} \tag{4.32}$$

Abbildung 4.12 zeigt ein Beispiel für eine Unterscheidung von reinen Unterschieden mit der Möglichkeit einer automatischen Konfliktlösung und Konflikten, die einen manuellen Eingriff erfordern. In dem aufgeführten Beispiel sind die resultierenden Status bei der Detektion von Unterschieden vermerkt. Konflikte, die nicht automatisch behoben werden können, sind mit einem Kreuz markiert. An dieser Stelle ist anzumerken, dass es ebenso

möglich wäre, an diesen Stellen einen resultierenden Status anzugeben, wenn dies der jeweilige Anwendungsfall zulässt.

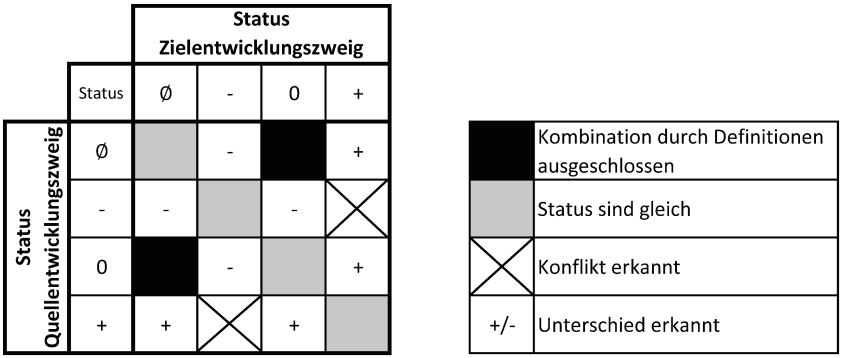


Abbildung 4.12: Beispiel für Umsetzung der Konfliktbeschreibungsmatrix mit Konfliktlösung (in Anlehnung an [HGU16, S. 7])

In Anlehnung an [HGU16] kann auf Basis der vorangegangenen mathematischen Definitionen die Erkennung von Unterschieden beziehungsweise Konflikten definiert werden. Hierfür werden im ersten Schritt alle Elemente herausgefiltert, die in den beiden zu vergleichenden Historien den gleichen Status aufweisen. Die resultierenden Mengen $\mathcal{D}_s$ und $\mathcal{D}_t$ sowie deren Berechnungsvorschrift ist in Gleichung 4.33 beschrieben.

$$\begin{aligned} \mathcal{D}_s &= \Omega_{End_s} \backslash (\Omega_{End_s} \cap \Omega_{End_t}) \\ \mathcal{D}_t &= \Omega_{End_t} \backslash (\Omega_{End_s} \cap \Omega_{End_t}) \end{aligned} \tag{4.33}$$

Die Kardinalität der beiden Mengen $\mathcal{D}_s$ und $\mathcal{D}_t$ kann unterschiedlich sein. Zur Verringerung der Komplexität der notwendigen mathematischen Operationen werden daher beide Mengen auf die gleiche Kardinalität erweitert. Dies ist möglich, indem jede der Mengen um die Elemente erweitert wird, die nur in der anderen Menge enthalten sind. Als Status wird dabei $\emptyset$ vergeben. Die formale Erweiterung ist in Gleichung 4.34 dargestellt.

$$\begin{aligned} \tilde{\mathcal{D}}_s = \{(s,\emptyset) \in (\mathcal{S} \times \mathcal{K}) | (s,g) \in \mathcal{D}_t \wedge (s,h) \notin \mathcal{D}_s;\ s \in \mathcal{S};\\ g \in \mathcal{K};\ h \in \mathcal{K};\ g\ beliebig;\ h\ beliebig\} \cup \mathcal{D}_s \\ \tilde{\mathcal{D}}_t = \{(s,\emptyset) \in (\mathcal{S} \times \mathcal{K}) | (s,g) \in \mathcal{D}_s \wedge (s,h) \notin \mathcal{D}_t;\ s \in \mathcal{S};\\ g \in \mathcal{K};\ h \in \mathcal{K};\ g\ beliebig;\ h\ beliebig\} \cup \mathcal{D}_t \end{aligned} \tag{4.34}$$

Durch die Erweiterung der Mengen auf die gleiche Kardinalität kann im Folgenden die Gleichung 4.35 angewendet werden. Bei dieser handelt es sich um die Zusammenführung der Mengen $\tilde{\mathcal{D}}_s$ und $\tilde{\mathcal{D}}_t$ zu einer gemeinsamen Menge, die aus Tripeln besteht. Diese Tripel

beinhalten an erster Stelle das Element, gefolgt von den beiden Status des Elementes auf dem Quell- und dem Zielentwicklungszweig. Auf dieser Ergebnismenge können dann wiederum Regelsätze angewendet werden, wie sie bereits beispielhaft anhand der Konfliktbeschreibungsmatrix aufgeführt wurden, um aus den Unterschieden entsprechende Ergebnismengen für die Zusammenführung abzuleiten.

$$\mathcal{D}_{Diff} = \{(s,k_s,k_t) \in (\mathcal{S} \times \mathcal{K} \times \mathcal{K})\} | (s,k_s) \in \tilde{\mathcal{D}}_s \wedge (s,k_t) \in \tilde{\mathcal{D}}_t\} \tag{4.35}$$

Der in Gleichung 4.32 beschriebene Regelsatz kann im Folgenden auf die Ergebnismenge aus Gleichung 4.35 angewendet werden, um eine automatische Konflikterkennung durchzuführen. Es resultiert die Menge $\mathcal{D}_Q$, die aus Quadrupeln besteht und in Gleichung 4.36 dargestellt ist. Diese beschreibt zu jedem Element s die zugehörigen Status, ob es sich um einen Konflikt handelt und was der resultierende Status ist.

$$\begin{aligned}\mathcal{D}_Q = \{(s,k_s,k_t,q,k_r) | (k_{s_Q},k_{t_Q},q,k_{r_q}) \in \mathcal{Q};\\ (s,k_s,k_t) \in \mathcal{D}_{Diff};\ k_s = k_{s_Q};\ k_t = k_{t_Q}\}\end{aligned} \tag{4.36}$$

Nachvollziehung von transienten Effekten auf High-Level-Ebene Neben transienten Effekten auf atomarer Änderungsebene können solche Effekte auch auf High-Level-Ebene vorkommen. Unter anderem können auf Basis von High-Level-Änderungen in vorangegangenen Revisionen wiederum Änderungen durchgeführt werden, die Einfluss auf eine anschließende Konflikterkennung und -behebung haben können, da resultierende Konflikte auf atomarer Ebene nicht festgestellt werden können. Für die Analyse der High-Level-Changes müssen diese entlang der zusammenzuführenden Pfade im ersten Schritt aus den atomaren Änderungen heraus aggregiert werden. Demzufolge ist es wiederum notwendig, eine gemeinsame Vorgängerrevision und die resultierenden Pfade zu generieren. Hierfür können die bereits in Gleichung 4.26 und Gleichung 4.27 eingeführten Definitionen als Ausgangspunkt wiederverwendet werden.

Basierend auf den vorliegenden Pfaden C_{path_s} und C_{path_t} wird die Aggregation vorgenommen. Gleichung 4.37 beschreibt das notwendige Vorgehen. Den in den Pfaden beschriebenen Änderungen werden, wenn möglich, High-Level-Changes zugeordnet. Da diese nach Gleichung 4.18 keine Zuordnung zu den korrespondierenden Revisionen mehr besitzen, werden diese jeder Aggregation zugeordnet, was eine nachträgliche Identifikation ermöglicht.

$$\begin{aligned}&\boldsymbol{hlcPathAgg_{\mathcal{G}}}\,(C_{path}) :=\\ &\left\{ (r_{x_n}, r_{y_n}, \boldsymbol{hlcAgg_{\mathcal{G}}}\,(r_{x_n}, r_{y_n})) \;\middle|\; \begin{array}{l} \mathcal{G} = (R_g, C_g, B_g, T_g, n_g);\\ n \in \mathbb{N}_0^+; 0 \leq n \leq |C_{path}| - 1;\\ (r_{x_n}, r_{y_n}, C_n^+, C_n^-) \in C_{path} \end{array} \right\}\end{aligned} \tag{4.37}$$

Im Folgenden werden durch Nutzung der Gleichung 4.37 Überschneidungen zwischen den erkannten High-Level-Changes detektiert. Gleichung 4.38 stellt die zugehörigen Vorschriften dar. Eine Abhängigkeit beziehungsweise eine aufeinander aufbauende Weiterentwicklung zwischen zwei High-Level-Changes wird erkannt, wenn es eine Schnittmenge zwischen der Menge der hinzugefügten Elemente und der Menge der gelöschten Elemente, der zu prüfenden High-Level-Changes, gibt. Ergebnis der Funktion $\boldsymbol{intersec}_{\mathcal{G}}(C_{path})$ ist eine Menge an Tupeln, die die aufgedeckten Abhängigkeiten der High-Level-Changes beschreibt. Für die Beschreibung werden Tupelbeziehungen genutzt, aus denen wiederum ein Abhängigkeitsbaum abgeleitet werden kann.

$$
\boldsymbol{intersec}_{\mathcal{G}}(C_{path}) :=
\left\{
\begin{array}{c|l}
 & \mathcal{G} = (R_g,C_g,B_g,T_g,n_g); \\
 & a \in \boldsymbol{hlcPathAgg}_{\mathcal{G}}(C_{path}); \\
 & b \in \boldsymbol{hlcPathAgg}_{\mathcal{G}}(C_{path});\ a \neq b; \\
 & a = \left(r_{x_a},r_{y_a},\left(\boldsymbol{\Phi}_{\mathcal{G}}(r_{x_a},r_{y_a}),C^{+}_{r_a},C^{-}_{r_a}\right)\right); \\
((r_{x_a},r_{y_a},h_a), & b = \left(r_{x_b},r_{y_b},\left(\boldsymbol{\Phi}_{\mathcal{G}}(r_{x_b},r_{y_b}),C^{+}_{r_b},C^{-}_{r_b}\right)\right); \\
(r_{x_b},r_{y_b},h_b)) & h_a \in \boldsymbol{\Phi}_{\mathcal{G}}(r_{x_a},r_{y_a}); \\
 & h_b \in \boldsymbol{\Phi}_{\mathcal{G}}(r_{x_b},r_{y_b}); \\
 & h_a = \left(C^{+}_{sub_{n_{za}}},C^{-}_{sub_{n_{za}}},n_{za}\right); \\
 & h_b = \left(C^{+}_{sub_{n_{zb}}},C^{-}_{sub_{n_{zb}}},n_{zb}\right); \\
 & \left(C^{+}_{sub_{n_{za}}} \cap C^{-}_{sub_{n_{zb}}}\right) \cup \left(C^{-}_{sub_{n_{za}}} \cap C^{+}_{sub_{n_{zb}}}\right) \neq \emptyset
\end{array}
\right\}
\tag{4.38}
$$

Erkennung von High-Level-Konflikten Die Erkennung von High-Level-Konflikten ist von dem verfügbaren Regelsatz für die Erkennung von High-Level-Changes und zusätzlichem Wissen über mögliche Konflikte beim Auftreten von Änderungen auf unterschiedlichen Entwicklungszweigen abhängig. Da es sich dabei um technologiespezifisches Wissen handelt, wird an dieser Stelle nicht auf eine formale Beschreibung eingegangen. Auf Basis der Aufbereitung der strukturellen Änderungen mithilfe der Gleichungen 4.18, 4.37 und 4.38 sind jedoch unterschiedliche Szenarien für die Konflikterkennung möglich, wie unter anderem:

- Analyse entlang des Pfades hat keine Abhängigkeiten zwischen den High-Level-Changes festgestellt:
  - Konflikterkennung ist auf der Ebene der einzelnen High-Level-Changes möglich
- Analyse entlang des Pfades hat Abhängigkeiten zwischen den High-Level-Changes festgestellt:
  - Konflikterkennung ist auf der Ebene der einzelnen High-Level-Changes innerhalb der Pfade möglich
  - Konflikterkennung ist auf der Ebene von Sequenzen von abhängigen High-Level-Changes innerhalb der Pfade möglich
  - Konflikterkennung ist auf der Ebene von Sequenzen von abhängigen High-Level-Changes und einzelnen High-Level-Changes innerhalb der Pfade möglich
  - Konflikterkennung ist auf der Ebene der Gesamtpfade möglich.

Es existieren weitere Szenarien, die die Kombination der bereits aufgeführten Möglichkeiten unterstützten. Hierzu gehört beispielsweise die Zusammenführung von einem Entwicklungszweig, der keine Abhängigkeiten aufweist, mit einem Entwicklungszweig, der mehrere Abhängigkeitspfade besitzt.

Bei dem Vergleich von Entwicklungszweigen auf einer High-Level-Ebene müssen Korrespondenzen zwischen den zu vergleichenden High-Level-Changes gefunden werden. Diese Korrespondenzen sind einerseits aus den Abhängigkeitspfaden ableitbar, wobei ausgehend von der gemeinsamen Ausgangsrevision eine Analyse entlang der Pfade durchgeführt werden kann. Andererseits ist es ebenfalls möglich, ein charakteristisches Element für eine jede High-Level-Change-Aggregationsregel zu definieren. Mit diesem Element kann dann ein Vergleich von aggregierten High-Level-Changes vom gleichen Typ durchgeführt werden. Beim Vergleich von Abhängigkeitssequenzen können des Weiteren Invertierungsbeziehungen zwischen High-Level-Changes beachtet werden, wodurch zusätzliche Konfliktmöglichkeiten ausgeschlossen werden können.

Konfliktbehebung Die Konfliktbehebung kann auf Basis, der in den vorangegangenen Abschnitten beschriebenen Methoden zur Erkennung von Unterschieden und Konflikten bei der Zusammenführung von Entwicklungszweigen durchgeführt werden. Im Wesentlichen bestehen zwei Möglichkeiten, um einen Konflikt zu beheben. Im ersten Fall muss ein Nutzer den Konflikt auf Basis der vorliegenden Information beheben. Im zweiten Fall kann durch die Analyse der Unterschiede und der bereitgestellten Regelsätze automatisch eine Konfliktlösung abgeleitet werden. Für eine automatische Konfliktlösung kann beispielsweise eine Priorisierung auf Nutzer- oder Entwicklungszweigebene erfolgen, um daraus die resultierenden Schritte für die Zusammenführung abzuleiten.

4.4.3.3 Semantische Beschreibung

Die in Abbildung 4.9 aufgeführte semantische Beschreibung für die Basisrevisionskontrollfunktionalitäten wird im Folgenden um die Möglichkeiten der Zusammenführung erweitert (Anforderung A-204). Abbildung 4.13 zeigt das zugehörige UML-Modell.

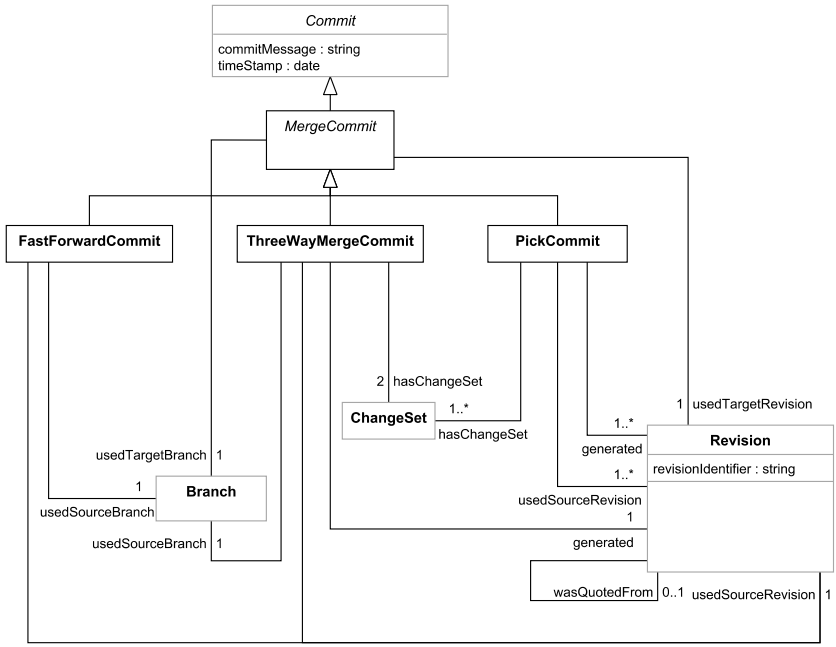


Abbildung 4.13: Semantische Beschreibung der Mergefunktionalitäten als UML-Modell (erweitert Abbildung 4.9)

Die semantische Beschreibung der im Abschnitt 4.4.3.1 eingeführten Methoden erfolgt durch Ableitung vom bereits existierenden *Commit*. Jeder *MergeCommit* hat dabei eine Referenz zu dem genutzten Zielentwicklungszweig (*usedTargetBranch*), auf den die Änderungen angewendet werden. Des Weiteren wird jeweils eine Zielrevision (*usedTargetRevision*) angegeben, die den Ausgangspunkt für die Anwendung der Änderungen darstellt. Für 3-Wege-Merges und Fast Forwards wird jeweils eine Quellrevision (*usedSourceRevision*) angegeben. Ebenso wird immer nur eine neue Revision mit dem zusammengeführten Inhalt generiert. Bei einem Pick können hingegen mehrere Quellrevisionen spezifiziert werden, wodurch demzufolge auch mehrere generierte Revisionen entstehen. Hieraus resultiert gleichzeitig auch die Notwendigkeit, die Beziehung zwischen der Ausgangsrevision und der erzeugten, kopierten Revision darzustellen. Dies erfolgt mittels der Assoziation *wasQuotedFrom*. Für 3-Wege-Merges müssen aufgrund der Zusammenführung von zwei Entwicklungszweigen auch zwei *ChangeSet*s angelegt werden, um die Historie entlang beider Zweige darstellen zu können. Ein Pick kann je nach Anzahl der kopierten Revisionen mehrere *ChangeSet*s referenzieren. Da ein Pick auf Revisionen von unterschiedlichen Entwicklungszweigen zugreifen kann, besitzt dieses

keine Referenz auf den Quellentwicklungszweig (*usedSourceBranch*), wie Fast Forwards und 3-Wege-Merges.

Abbildung 4.14 erweitert die bestehenden Abbildungen 4.9 und 4.10 um die semantische Beschreibung der Konflikterkennung und -behebung. Hierfür wird eine neue Klasse *Difference* eingeführt, die kennzeichnet, ob es sich um einen konfliktbehafteten Unterschied handelt oder nicht. Es stehen zwei mögliche Ausprägungen zur Verfügung. Strukturelle Unterschiede können mittels *StructuralDifference* und High-Level-Unterschiede können mittels *HighLevelDifference* beschrieben werden. Strukturelle Unterschiede sind durch *sourceState* (Status in Quellentwicklungszweig), *targetState* (Status in Zielentwicklungszweig), *ResolutionState* (Status für eine regelbasierte Konfliktlösung) und *correspondingStatement* (Element auf das sich der Unterschied bezieht) gekennzeichnet.

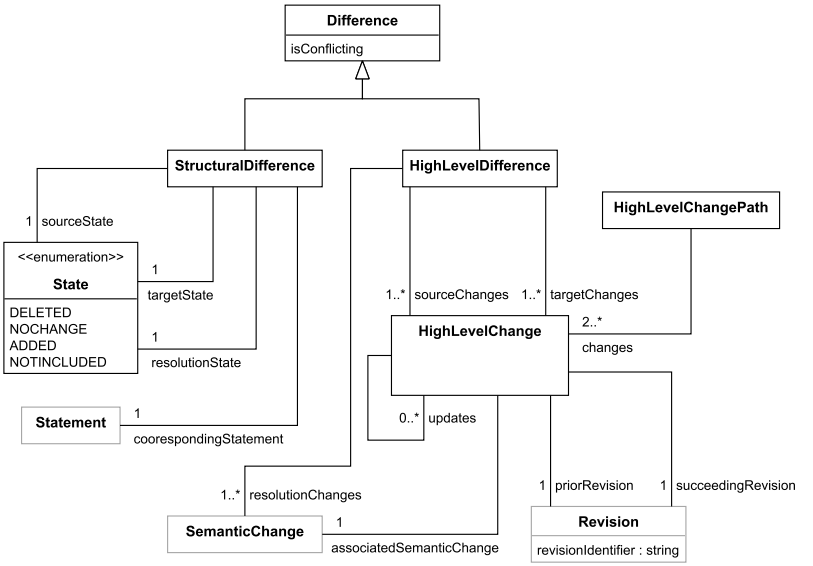


Abbildung 4.14: Semantische Beschreibung der Konflikterkennung und -behebung als UML-Modell (erweitert Abbildungen 4.9 und 4.10)

Die einzelnen Status beziehen sich auf eine Enumeration an Möglichkeiten, die die Status aus Gleichung 4.25 widerspiegeln. High-Level-Unterschiede setzen sich aus einer Menge von High-Level-Changes (*HighLevelChange*) in Bezug auf Quell- und Zielentwicklungszweig zusammen. Des Weiteren kann eine Konfliktlösung spezifiziert werden, die unter Umständen auch aus mehreren *SemanticChange*s bestehen kann. Jedem *HighLevelChange* ist ein korrespondierender *SemanticChange* sowie eine Vorgänger- (*priorRevision*) und eine Nachfolgerrevision (*succeedingRevision*) zugeordnet. Mittels der Assoziation *updates* können die Abhängigkeiten zwischen den einzelnen *HighLevelChange*s innerhalb eines

Pfades beschrieben werden. Ein *HighLevelChangePath* besteht aus mindestens zwei *HighLevelChange*s.

4.5 Evolutions- und Konsistenzmechanismen

In diesem Abschnitt erfolgt die Erweiterung des technologieunabhängigen Revisionskontrollsystems um Evolutions- und Konsistenzmechanismen, die für die Realisierung eines RMS notwendig sind. Grundlage bilden hierfür die mathematischen und semantischen Beschreibungen aus den vorangehenden Abschnitten. Insbesondere wird auf den Basisrevisionskontrollfunktionalitäten, der Aggregation von High-Level-Changes und der formalen Beschreibung verbindungsorientierter Modelle aufgesetzt. Die nachfolgenden Definitionen beschreiben hierbei wiederum ein Integrationsschema, in das unterschiedliche technologiespezifische Regelsätze und Algorithmen zur Durchführung von Evolutionen, wie zum Beispiel in [Keh15], [Sto04], [Pap+13] oder [AH06] dargestellt, integriert werden können. Für die semantische Beschreibung werden UML-Modelle verwendet, wobei bereits in vorhergehenden Abschnitten definierte Elemente wiederum grau hervorgehoben werden.

4.5.1 Evolutionsmechanismen

Ausgehend von einem Beispiel der Co-Evolution innerhalb des RMS wird im Folgenden die notwendige Revisionsgraphenstruktur vorgestellt. Anschließend werden die mathematischen sowie semantischen Beschreibungen für die Co-Evolution dargelegt (Anforderung A-301).

4.5.1.1 Integration in RMS

Für die Durchführung von Co-Evolutionen innerhalb des RMS wird auf die Basisrevisionskontrollfunktionalitäten zurückgegriffen. Abbildung 4.15 zeigt ein Beispiel für die Strukturierung der beteiligten Revisionsgraphen innerhalb des RMS. Darin wird davon ausgegangen, dass zwei voneinander abhängige Modelle in unterschiedlichen Revisionsgraphen im Revisionsverwaltungssystem existieren. Die Verbindungen zwischen den Modellen werden wiederum in einem weiteren Revisionsgraphen gespeichert. Des Weiteren existiert ein Revisionsgraph, der durchgeführte Co-Evolutionen dokumentiert und ein Revisionsgraph mit Regelsätzen, die zum Beispiel für die Aggregation von High-Level-Changes oder die Anwendung von Co-Evolutionen verwendet werden können.

Durch die Evolution des Revisionsgraphen von *Modell 1* wird automatisiert analysiert, ob eine Co-Evolution erforderlich ist. Hierfür werden die im Revisionsgraph *Verbindungen* vorgehaltenen Verbindungen analysiert, um abhängige Revisionsgraphen zu identifizieren. In dem dargestellten Beispiel existieren Abhängigkeiten zwischen *Modell 1* und *Modell 2*. Auf Basis der Aggregation der durchgeführten Änderungen am *Modell 1* wird unter Nutzung der *Regelsätze* eine notwendige Co-Evolution durchgeführt. Diese resultiert in einer neuen Revision von *Modell 2* und der Erzeugung einer neuen Revision im Revisionsgraphen *Co-Evolution*, um die durchgeführten Änderungen semantisch zu beschreiben. Hierzu gehören Referenzen auf die bisherigen sowie die neuen Stände von

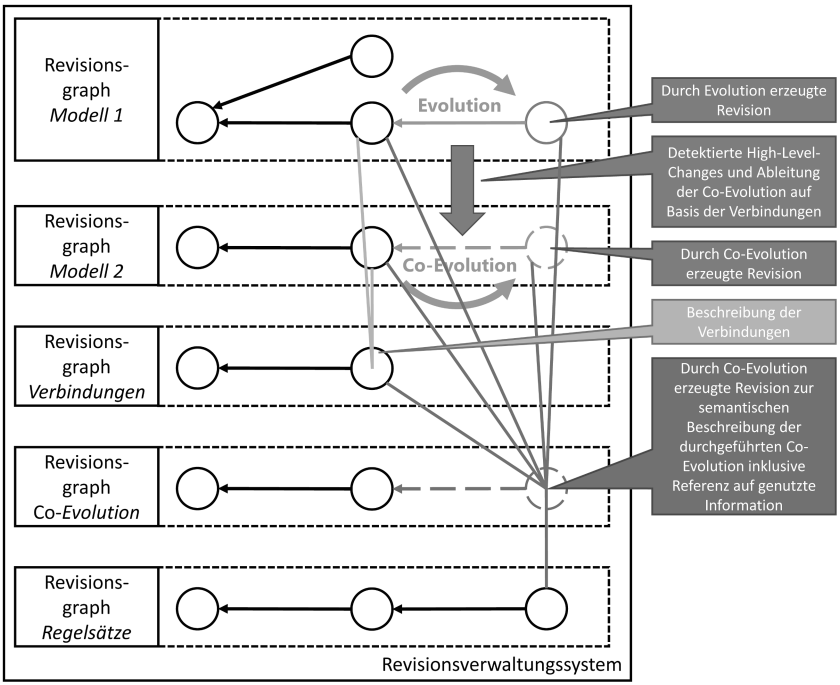


Abbildung 4.15: Beispiel für die Umsetzung einer Co-Evolution innerhalb des RMS

Modell 1 und *Modell 2*, die genutzten Verbindungen sowie die angewendeten Regelsätze für Aggregation und Co-Evolution.

Generell können innerhalb des RMS unterschiedliche Strukturierungen der Revisionsgraphen vorgenommen werden. So können zum Beispiel Informations- und Datenmodelle innerhalb eines oder in unterschiedlichen Revisionsgraphen vorgehalten werden. Für die Beschreibung der Verbindungen kann entweder ein eigener Revisionsgraph verwendet werden, wodurch zum Beispiel Abhängigkeiten zwischen unterschiedlichen Datenmodellen gepflegt werden können. Es ist jedoch ebenso möglich, dass die Verbindungen direkt in den Datenmodellen enthalten sind. So können unter anderem Typ-Instanzbeziehungen direkt in den Datenmodellen abgebildet werden, indem auf die zugehörigen Informationsmodelle referenziert wird. Die Co-Evolution wird innerhalb des RMS jeweils in einem eigenen Revisionsgraphen dokumentiert, da hierdurch Eingriffe in die zu pflegenden Modelle vermieden werden und die Co-Evolution eine eigene Revisionshistorie besitzt. Regelsätze sollten mit einem Verweis auf den genutzten Stand ebenfalls semantisch referenzierbar sein.

4.5.1.2 Mathematische Beschreibung

Für die im vorangegangenen Abschnitt eingeführte semantische Beschreibung der durchgeführten Co-Evolution wird ein entsprechender Revisionsgraph $\mathcal{G}_e$ benötigt. Gleichung 4.39 gibt die zugehörige Vorschrift für die Erstellung an. Der resultierende Revisionsgraph beinhaltet nur einen *master*-Zweig, auf dem die Historie der Co-Evolution beschrieben wird.

$$\boldsymbol{create}_{\Gamma}(\emptyset) = \Gamma \cup \{\mathcal{G}_e\} = \Gamma' \tag{4.39}$$

Grundlage für die Durchführung einer Co-Evolution sind Änderungen an einem Modell, das Abhängigkeiten zu anderen Modellen aufweist. Daher wird im Folgenden angenommen, dass bereits ein oder mehrere Commits auf einen entsprechenden Revisionsgraphen durchgeführt wurden. Darauf aufbauend kann die Funktion $\boldsymbol{coevolveAll}_{\Gamma}(\mathcal{G}_s,r_{x_s},r_{y_s})$ genutzt werden, um die Änderungen in Form von Co-Evolutionen an alle abhängigen Modelle in Γ zu propagieren. Bei dieser Funktion handelt es sich um eine aufeinanderfolgende Ausführung von mehreren Funktionen. Zu den verwendeten Funktionen gehören auf der einen Seite Basisrevisionskontrollfunktionalitäten, wie in den vorangehenden Abschnitten beschrieben, aber auf der anderen Seite auch spezifische Funktionen der Co-Evolution. Des Weiteren müssen Parameter für die Ausführung der Funktion spezifiziert werden. Dazu zählen der Quellrevisionsgraph $\mathcal{G}_s$ und die Startrevision r_{x_s} und Endrevision r_{y_s}, zwischen denen die durchgeführten Änderungen untersucht werden sollen. Die zu co-evolvierenden Änderungen werden im ersten Schritt zu High-Level-Changes aggregiert. Hierfür wird die Funktion $\boldsymbol{hlcAgg}_{\mathcal{G}}$ aus Gleichung 4.18 verwendet. Als Parameter erhält diese den Quellrevisionsgraph sowie Start- und Endrevision. Das Ergebnis der Aggregation wird in der Variablen $h_{\mathcal{G}_s}$ gespeichert. Die Abhängigkeit der durchgeführten Änderungen wird unter Nutzung von $h_{\mathcal{G}_s}$ und den beschriebenen Verbindungen in Revisionsgraphen von Γ abgeleitet. Die Funktion $\boldsymbol{calcDep}_{\Gamma}(\mathcal{G}_s,r_{x_s},r_{y_s},h_{\mathcal{G}_s})$ stellt die dafür benötigte Funktionalität bereit. Die Realisierung ist dabei wiederum vom konkreten Anwendungsfall und den zugrunde liegenden Regelsätzen für die Erkennung von Abhängigkeiten abhängig. Abhängigkeiten können dabei sowohl auf einer sehr abstrakten Ebene oder auf die Domäne zugeschnitten spezifiziert werden. Die generelle Definition der Funktion mit der zugehörigen Ergebnismenge ist in Gleichung 4.40 aufgeführt. Die Ergebnismenge setzt sich aus Tupeln zusammen, die abhängige Entwicklungszweige und den zugehörigen Revisionsgraphen angeben.

$$\boldsymbol{calcDep}_{\Gamma}(\mathcal{G}_s,r_{x_s},r_{y_s},h_{\mathcal{G}_s}) := \left\{ (\mathcal{G}_t,n_{b_t}) \;\middle|\; \begin{array}{l} \mathcal{G}_t \in \Gamma; \\ \mathcal{G}_t = (R_{g_t},C_{g_t},B_{g_t},T_{g_t},n_{g_t}); \\ b_t \in B_{g_t}; \\ b_t = (R_{b_t},r_{l_t},\Upsilon_{l_t},n_{b_t}) \end{array} \right\} \tag{4.40}$$

Die eigentliche Durchführung der Co-Evolution erfolgt mittels entsprechender Commits auf die abhängigen Entwicklungszweige der jeweiligen Revisionsgraphen. Hierfür steht die in Gleichung 4.41 definierte Funktion $\boldsymbol{coevolve}_{\Gamma}\left(h_{\mathcal{G}_s},\mathcal{G}_t,n_{b_t}\right)$ zur Verfügung. Diese aktualisiert den Zielrevisionsgraphen, indem eine neue Revision auf dem Zielentwicklungszweig angelegt wird, in der die notwendigen Änderungen zur Umsetzung der Co-Evolution durchgeführt werden. Die Berechnung der zugehörigen Hinzufügungen und Löschungen wird mittels $\mathcal{E}\left(h_{\mathcal{G}_s},\mathcal{G}_t,n_{b_t}\right)$, beschrieben in Gleichung 4.42, vorgenommen. Die Realisierung basiert dabei auf den Regelsätzen und Algorithmen der Co-Evolution, wie zum Beispiel in [Keh15] oder [Sto04] beschrieben. Für die Integration in das Gesamtsystem werden dabei die bereits detektierten High-Level-Changes als Grundlage für die Umsetzung der Regelsätze verwendet.

$$\begin{aligned}\boldsymbol{coevolve}_{\Gamma}\left(h_{\mathcal{G}_s},\mathcal{G}_t,n_{b_t}\right) &:= \left(\Gamma\setminus\{\mathcal{G}_t\}\right)\cup\left\{\boldsymbol{commit}_{\mathcal{G}_t}\left(n_{b_t},\mathcal{E}\left(h_{\mathcal{G}_s},\mathcal{G}_t,n_{b_t}\right)\right)\right\}\\ &= \Gamma'\end{aligned} \tag{4.41}$$

$$\mathcal{E}\left(h_{\mathcal{G}_s},\mathcal{G}_t,n_{b_t}\right) = \left(C^{+},C^{-}\right) \tag{4.42}$$

Gleichung 4.43 fasst den Ablauf der Funktion $\boldsymbol{coevolveAll}_{\Gamma}\left(\mathcal{G}_s,r_{x_s},r_{y_s}\right)$ zusammen. Nach der High-Level-Change-Aggregation wird eine leere Menge $\mathcal{Z}$ definiert, die für die temporäre Speicherung der durchgeführten Änderungen verantwortlich ist. In der folgenden Schleife werden für alle gefundenen Abhängigkeiten Co-Evolutionen durchgeführt und jeweils der entsprechende Revisionsgraph durch einen Commit aktualisiert. Die durchgeführte Änderung mit allen genutzten Variablen wird dann zu $\mathcal{Z}$ hinzugefügt. Die Variable r_{l_t} gibt hierbei die neu erstellte Revision an, die identisch zum Blatt des Entwicklungszweiges b_t ist. Der Entwicklungszweig b_t wird mittels n_{b_t} identifiziert. Nachdem die Co-Evolution für alle Abhängigkeiten durchgeführt wurde, wird $\mathcal{G}_e$ aktualisiert, wofür $\mathcal{Z}$ die Grundlage bildet. Diese Menge der durchgeführten Änderungen wird mittels der Funktion $\boldsymbol{\Xi}(\mathcal{Z})$ in die semantische Beschreibung mit den zugehörigen Hinzufügungen und Löschungen überführt. Alle Änderungen werden dabei auf den *master*-Zweig geschrieben.

$\boldsymbol{coevolveAll}_{\Gamma}\left(\mathcal{G}_s,r_{x_s},r_{y_s}\right)$:

$$\begin{array}{ll}
1: & h_{\mathcal{G}_s} = \boldsymbol{hlcAgg}_{\mathcal{G}_s}\left(r_{x_s},r_{y_s}\right) = \left(\boldsymbol{\Phi}_{\mathcal{G}_s}\left(r_{x_s},r_{y_s}\right),C_r^{+},C_r^{-}\right);\\
2: & \mathcal{Z} = \emptyset;\\
3: & \text{FOR } \left(\mathcal{G}_t,n_{b_t}\right) \text{ IN } \boldsymbol{calcDep}_{\Gamma}\left(\mathcal{G}_s,r_{x_s},r_{y_s},h_{\mathcal{G}_s}\right):\\
4: & \quad \Gamma = \boldsymbol{coevolve}_{\Gamma}\left(h_{\mathcal{G}_s},\mathcal{G}_t,n_{b_t}\right);\\
5: & \quad \mathcal{Z} = \mathcal{Z}\cup\left\{\left(\mathcal{G}_s,r_{x_s},r_{y_s},h_{\mathcal{G}_s},\mathcal{G}_t,n_{b_t},r_{l_t}\right)\right\};\\
6: & \Gamma = \left(\Gamma\setminus\{\mathcal{G}_e\}\right)\cup\left\{\boldsymbol{commit}_{\mathcal{G}_e}\left(1,\boldsymbol{\Xi}(\mathcal{Z})\right)\right\};
\end{array} \tag{4.43}$$

4.5.1.3 Semantische Beschreibung

Das in Abbildung 4.9 dargestellte UML-Modell zur Beschreibung der Revisionskontrollfunktionalitäten bildet im Folgenden die Grundlage für die Beschreibung der durchgeführten Co-Evolutionen (Anforderung A-302). Im ersten Schritt wird zur Beschreibung der Co-Evolutionsrevisionsgraphen und möglicher Revisionsgraphen für die Beschreibung von Verbindungen jeweils eine Ableitung vom *RevisionGraph* angelegt. Alle Co-Evolutionen in einem *RevisionControlSystem* werden dabei in einem Revisionsgraphen *CoEvolutionRevisionGraph* revisioniert. Des Weiteren kann eine beliebige Anzahl an Revisionsgraphen für die Verbindungsbeschreibung *ConnectionsRevisionGraph* pro Revisionskontrollsystem vorgehalten werden. Durch die explizite Beschreibung der Revisionsgraphtypen können diese den entsprechenden Komponenten des RMS für die Auswertung zugeordnet werden.

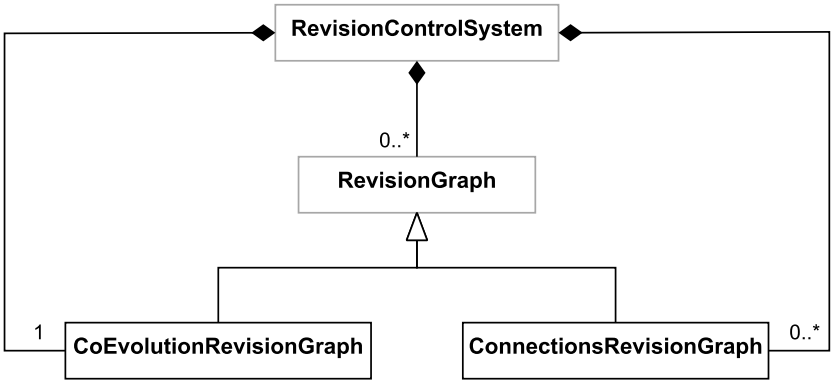


Abbildung 4.16: Semantische Beschreibung der Revisionsgraphen für Co-Evolution und Verbindungen

Die semantische Beschreibung der durchgeführten Co-Evolutionen erfolgt innerhalb des Co-Evolutionsrevisionsgraphen mithilfe des in Abbildung 4.17 dargestellten UML-Modells. Bei Änderungen an einem Modell innerhalb des Revisionsverwaltungssystems wird eine neue *Evolution* angelegt, die Referenzen auf den Quellrevisionsgraph (*usedSourceRevisionGraph*) und den zugehörigen Revisionsbereich, auf dem die Evolution aufbaut, besitzt. Der Revisionsbereich wird durch eine Startrevision (*startRevision*) und eine Endrevision (*endRevision*) beschrieben. Des Weiteren werden durch die *Evolution* die bereits aggregierten High-Level-Changes (*SemanticChange*) referenziert, auf deren Basis die späteren Co-Evolutionen durchgeführt werden (*associatedSemanticChange*). Die einzelnen Co-Evolutionen, die aus einer Evolution resultieren, werden mittels *CoEvolution* beschrieben (*performedCoEvolution*). Jede der Co-Evolutionen hat wiederum Referenzen auf den Zielrevisionsgraphen (*usedTargetRevisionGraph*) sowie dessen Zielentwicklungszweig (*usedTargetBranch*). Für die Identifikation dieses Ziels wurden eventuell explizit in einem *ConnectionsRevisionGraph* beschriebene Verbindungen genutzt. Wenn dies der Fall

ist, so werden die entsprechenden Revisionsgraphen mittels *usedConnectionsRevisionGraph* und den zugehörigen Revisionen (*usedConnectionsRevision*) referenziert, um eine Nachvollziehbarkeit der analysierten Abhängigkeiten sicherzustellen. Im Fall, dass keine Instanzen von *ConnectionsRevisionGraph* im Revisionskontrollsystem vorhanden sind, werden nur die in den Modellen beinhalteten Abhängigkeiten analysiert. Die Nachvollziehbarkeit ist auch in diesem Fall gewährleistet, da auf den Zielentwicklungszweig und die neu erstellte Revision (*generated*) verwiesen wird. Bei einer entsprechenden Analyse der Co-Evolution kann dann die Vorgängerrevision der neu erstellten Revision genutzt werden. Diese gibt den zugehörigen Stand an, der für die Co-Evolution als Grundlage genutzt wurde.

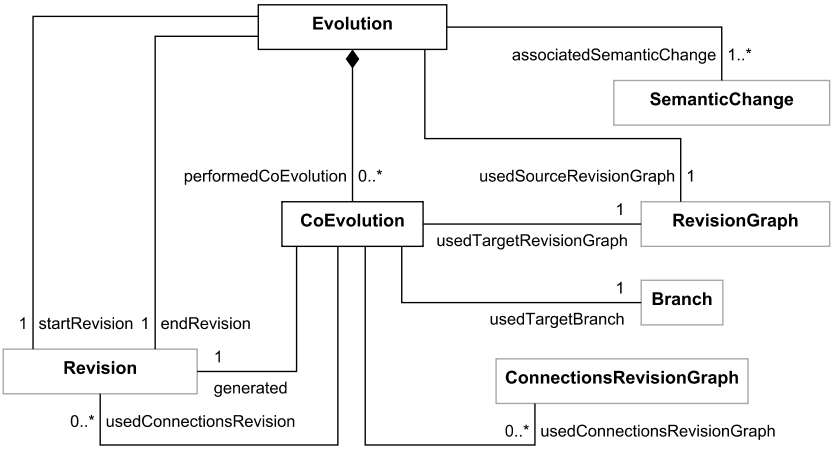


Abbildung 4.17: Semantische Beschreibung der Evolution als UML-Modell

4.5.2 Konsistenzmechanismen

Innerhalb des RMS muss die Konsistenz zwischen aber auch innerhalb von Modellen sichergestellt werden. Konsistenz liegt nach der in dieser Arbeit verwendeten Definition immer dann vor, wenn entsprechende Regelsätze für die Sicherung der Konsistenz eingehalten werden, was daher also auch im Fall einer Evolution sichergestellt werden muss. Einen ersten Anhaltspunkt bieten hierfür die detektierten High-Level-Changes. So kann bei Anwendung der Funktion $\boldsymbol{hlcAgg_{\mathcal{G}}}$ aus Gleichung 4.18 überprüft werden, ob sich die durchgeführten atomaren Änderungen vollständig zu High-Level-Changes aggregieren lassen können (Anforderung A-303). Im Fall, dass C_r^+ oder C_r^- ungleich der leeren Menge ist, ist entweder die Güte der zugrunde liegenden Regelsätze nicht ausreichend oder es wurden Änderungen vorgenommen, die zu Inkonsistenzen führen. Diese Inkonsistenzen können in den unterschiedlichen Typen der Modellkonsistenz nach Lucas *et al.* [LMT09]

auftreten. Innerhalb des RMS sind daher nur Änderungen zugelassen, die konsistent zur vorgegebenen Syntax sind. Eine entsprechende Prüfung kann direkt bei der Ausführung des Commits mittels $\boldsymbol{commit_{\mathcal{G}}}$ durchgeführt und inkonsistente Anfragen können hieraus direkt abgelehnt werden. Sowohl die horizontale als auch die vertikale Konsistenz werden durch entsprechende Regelsätze zur Beschreibung von Verbindungen zwischen Modellen auf gleicher oder unterschiedlicher Abstraktion sichergestellt. Darauf aufbauend werden die Regelsätze zur Durchführung der Co-Evolution definiert. Hieraus ergibt sich, dass falls eine semantische Beschreibung einer Co-Evolution innerhalb des RMS vorliegt, eine Konsistenzsicherstellung zwischen den abhängigen Modellen stattgefunden hat. Diese muss daher nicht im semantischen Modell explizit abgebildet werden. Aufbauend auf den Regelsätzen zur Sicherstellung von vertikaler und horizontaler Konsistenz können anschließend erweiterte Regelsätze für die Sicherstellung der semantischen Konsistenz angewendet werden, die nach erfolgreicher Durchführung von Evolution und Co-Evolution Konsistenz prüfen können.

5 Implementierung

Die Umsetzung des technologieunabhängigen Entwurfs erfolgt in diesem Abschnitt beispielhaft für das Semantic Web, da hier auf der einen Seite bereits erste Vorarbeiten existieren, um Revisionsverwaltung in diesem Bereich zu etablieren. Zum anderen ist die Revisionierung von industriellen Informationsmodellen nach Graube [Gra16] unter Bezugnahme auf Schmidt *et al.* [Sch+14] eine der Hauptanforderungen im Engineering Prozess und spielt daher auch eine zentrale Rolle innerhalb von LED [Gra16].

Die Basistechnologie des Semantic Web ist das RDF. Dadurch kann die Menge $\mathcal{S}$, die in Abschnitt 4.4 eingeführt wurde, durch die Menge der Tripel ersetzt werden. Ein Tripel besteht innerhalb des RDF jeweils aus Subjekt, Prädikat und Objekt. Mathematisch lässt sich die resultierende Menge $\mathcal{S}$ durch $\mathcal{S} := \mathcal{U} \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L})$ beschreiben. Bei $\mathcal{U}$ und $\mathcal{L}$ handelt es sich um zwei disjunkte unendliche Mengen, wobei $\mathcal{U}$ die Menge aller Uniform Resource Identifier (URI)s und $\mathcal{L}$ die Menge aller Literale beschreibt. Ein RDF-Graph ist des Weiteren definiert als eine Teilmenge der möglichen Tripel.

Grundlage für die Umsetzung bildet das semantische Revisionsverwaltungssystem R43ples, das im Folgenden zu einem RMS weiterentwickelt wird. Zu Beginn wird ein Überblick über die interne Architektur von R43ples und den zugehörigen notwendigen Erweiterungen gegeben, bevor im Anschluss die umgesetzten Funktionalitäten detaillierter beschrieben werden.

5.1 Übersicht

R43ples ist in vorangegangenen Arbeiten bereits konzeptionell entwickelt und prototypisch implementiert worden [Hen13; Hen14]. Durch das System werden Basisrevisionsverwaltungsoperationen aber auch bereits erste Ansätze für die Zusammenführung von divergierten Entwicklungszweigen bereitgestellt. Im Rahmen dieser Arbeit erfolgt eine Weiterverwendung des existierenden Konzepts sowie der zugehörigen Implementierung hin zu einem RMS. Für die Erreichung dieses Ziels wird R43ples sowohl konzeptionell als auch implementierungstechnisch erweitert. Insbesondere werden die entwickelten formalen und semantischen Beschreibungen dieser Arbeit in R43ples überführt. Die Implementierung erfolgt dabei Open Source auf GitHub und ist unter `https://github.com/plt-tud/r43ples` abrufbar.

Innerhalb von R43ples wird ein objektorientierter Implementierungsansatz verfolgt. Im Rahmen dieser Arbeit wird dieser weiter intensiviert. So wird die Kapselung und die Wiederverwendbarkeit von Funktionalitäten durch zusätzliche Paketstrukturen und die Schaffung von Interfaces unterstützt. Im Wesentlichen erfolgt eine Unterteilung in Kernfunktionalitäten und darauf aufbauenden weiterführenden Funktionen. Die Kernfunktionalitäten sind dabei innerhalb eines Pakets gekapselt und über ein Interface abrufbar. Das in Abbildung 5.1 dargestellte UML-Diagramm zeigt einen Ausschnitt

aus der Struktur des *core*-Pakets, das die Funktionen unter anderem an den Webserviceendpunkt bereitstellt. Innerhalb des Kernpaketes werden vorrangig Entwürfe von Objekten, wie beispielhaft in Abbildung 5.2 dargestellt, erzeugt, die dann mittels einer entsprechenden Methode in eine semantische Beschreibung im angeschlossenen Triple Store überführt werden.

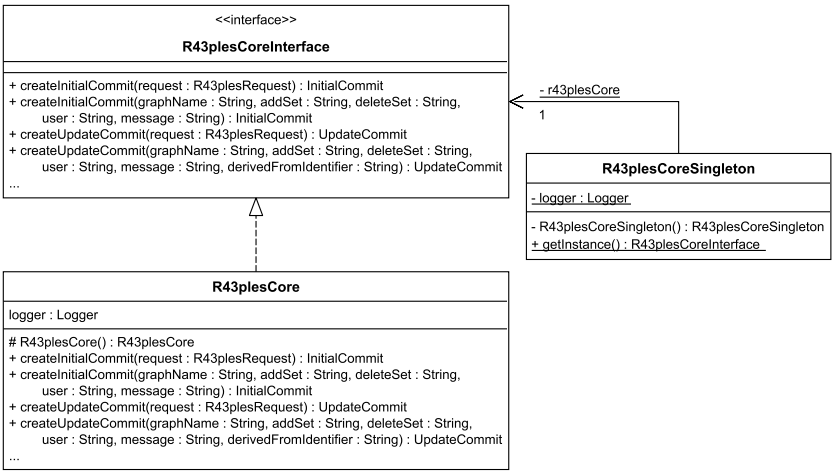


Abbildung 5.1: Ausschnitt aus der Struktur des *core*-Pakets als UML-Modell

Für den Aufruf der entsprechenden Funktion innerhalb des *R43plesCoreInterfaces* stehen in den meisten Fällen unterschiedliche Möglichkeiten bereit. So kann auf der einen Seite direkt eine Anfrage (*R43plesRequest*), die an den Webserviceendpunkt gestellt wurde, verarbeitet werden, beziehungsweise auf der anderen Seite ein parametrierter Aufruf gestartet werden. Letzterer wird dabei sowohl in der ersten Variante genutzt, wobei vorher eine Extraktion der Parameter aus der Anfrage vorgenommen wird, als auch innerhalb von abhängigen Funktionen, wenn beispielsweise ein neuer Commit aufgrund einer Co-Evolution erzeugt werden muss.

Für die leichtere Interaktion mit den semantischen Beschreibungen innerhalb des Programmcodes werden die bestehenden semantischen Beschreibungen in Objekte im Programmcode reflektiert. Hierbei ist zu beachten, dass der überwiegende Teil der notwendigen Information zu den Objekten nicht direkt ab der Erzeugung der Objekte vorgehalten wird, sondern bei Bedarf nachgeladen wird. Ein Ausschnitt der reflektierten Objekte ist in Abbildung 5.3 dargestellt.

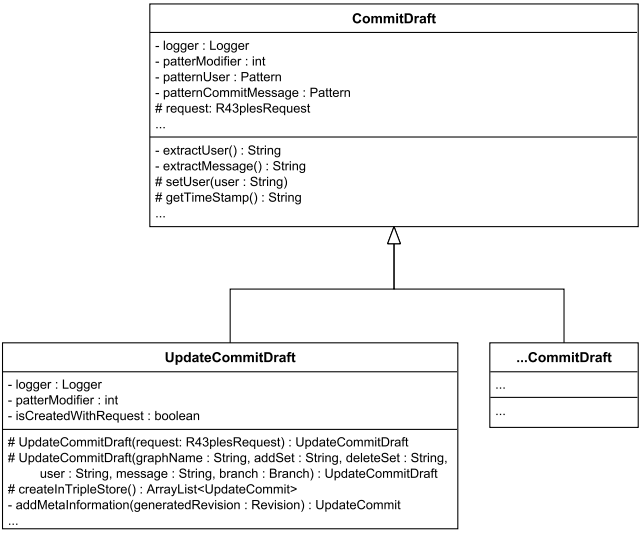


Abbildung 5.2: Ausschnitt aus den Entwurfsobjekten des *core*-Pakets als UML-Modell

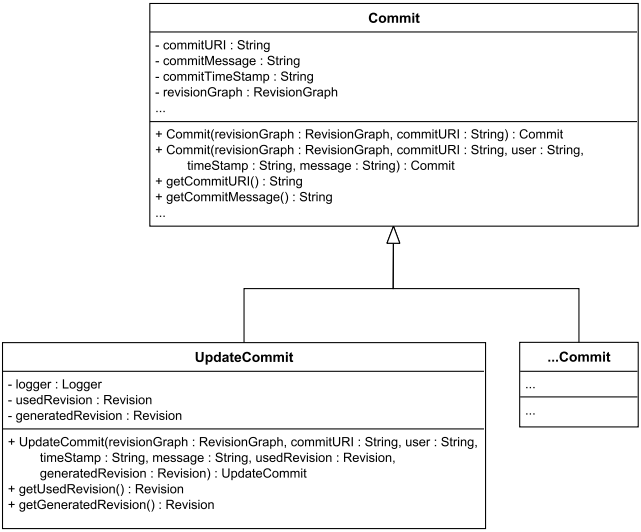


Abbildung 5.3: Ausschnitt der reflektierten Objekte als UML-Modell

5.2 Änderungsmanagement

Das Änderungsmanagement bildet, wie in Abschnitt 4.4 dargestellt, die Basis für den Aufbau weiterer Funktionalitäten und befindet sich daher im Kern von R43ples. Im Folgenden werden die Weiterentwicklung der semantischen Beschreibung innerhalb von R43ples sowie die erweiterten Anfrage- und Interaktionsmechanismen dargestellt.

5.2.1 Ontologie

Die semantische Beschreibung innerhalb von R43ples erfolgt mittels der Revision Management Ontology (RMO). Die aus den vorangegangenen Arbeiten hervorgegangene Beschreibung der Revisionsinformation ist in Abbildung 5.4 als UML-Modell der Ontologie dargestellt. Als Grundlage für die Modellierung wird dabei PROV-O[1)], sowie OWL und RDFS verwendet. Bei dieser Darstellung ist zu beachten, dass alle Attribute der Klassen, die im Namensraum von PROV-O liegen, nicht in der RMO abgebildet, aber für die Modellierung der Revisionsinformation verwendet werden. Die Beschreibung beinhaltet die Basisklassen für die Modellierung von Graphen, Revisionen, Commits und Referenzen. Commits werden dabei nicht weiter unterschieden. Die Zusammenführung von divergierten Entwicklungszweigen wird nicht semantisch dargestellt und kann nur auf Ebene des Zusammenhangs von Revisionen nachvollzogen werden. Die hinzugefügten und gelöschten Tripel werden nur in Bezug auf die erzeugte Revision beschrieben. Dies ist ausreichend, wenn davon ausgegangen wird, dass immer eine weiter in der Vergangenheit liegende Revision als Ausgangspunkt verwendet wird, wie in [Hen14] vorgestellt.

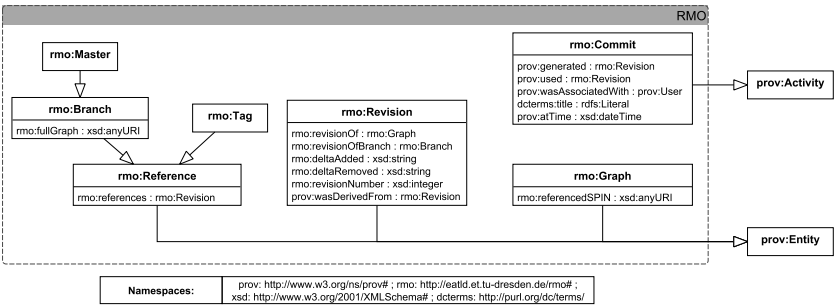


Abbildung 5.4: Bisheriger Stand der semantischen Beschreibung mittels der RMO als UML-Modell (in Anlehnung an [Hen13; Hen14; GHU14])

Im Fall, dass nur die letzte Revision eines Zweiges vollständig vorliegt und die beiden Revisionen vor der Zusammenführung wiederhergestellt werden sollen, ist es nicht möglich mit der Beschreibung aus [Hen14] den Ausgangszustand wiederherzustellen. Dies ist jedoch notwendig, wenn beispielsweise Algorithmen für die Optimierung der

[1)] http://www.qudt.org/ (besucht am 29.11.2020)

Wiederherstellung von einzelnen Revisionen eingesetzt werden, da diese unterschiedliche Pfade und Richtungen nutzen können, um den bestmöglichen Pfad zu finden.

Die RMO wird im Folgenden mittels den im Abschnitt 4.4 eingeführten semantischen Beschreibungen erweitert und verändert, wobei PROV-O weiterhin die Grundlage für die Modellierung bildet. Abbildung 5.5 zeigt die resultierende Ontologie als UML-Modell. Attribute der Klassen sind vollständig in die RMO überführt, wodurch eine vollständige semantische Beschreibung der Revisionsinformationsmodellierung vorliegt, ohne dass zusätzliches Wissen über die Verwendung von weiteren Ontologien notwendig ist. Die Attribute sind aber weiterhin zum großen Teil von PROV-O abgeleitet, was jedoch, wie auch die Multiplizitäten, aus Gründen der Übersichtlichkeit in dieser UML-Darstellung nicht abgebildet ist.

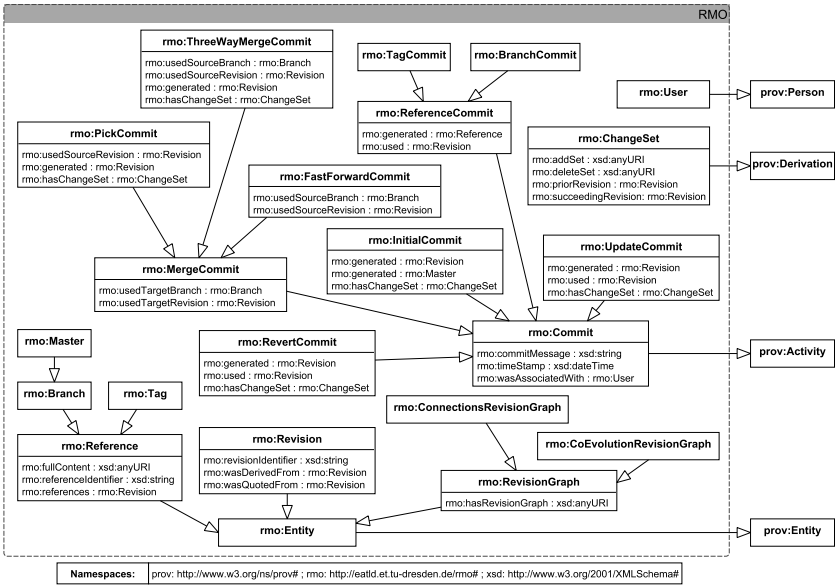


Abbildung 5.5: RMO nach der Erweiterung als UML-Modell

Für die unterschiedliche Commitarten existieren durch die Erweiterung separate Ableitungen, wodurch die Semantik der durchgeführten Änderungen nachvollzogen werden kann. Die Einführung des *rmo:ChangeSet*s zwischen zwei Revisionen behebt außerdem das im vorangegangenen Abschnitt erläuterte Problem bei der Wiederherstellung von Revisionsinhalten, wenn eine Zusammenführung entlang der Wiederherstellung durchgeführt wurde. Die in Abbildung 4.9 eingeführten Beschreibungen für das *rmo:Set* und die darin enthaltenen *rmo:Statement*s sind in der RMO nicht enthalten, da hierfür direkt RDF-Graphen für die Speicherung der Tripel genutzt werden können. Entsprechend wird

mittels *xsd:anyURI* auf die zugehörigen URIs der Graphen verwiesen. In den folgenden Abschnitten, werden die für die Aggregation und Co-Evolution notwendigen zusätzlichen Erweiterungen schrittweise ergänzt.

5.2.2 Basisrevisionskontrollfunktionalitäten

Die Zugriffsmöglichkeiten werden, wie bereits in [GHU14] dargestellt, mittels einer Erweiterung von SPARQL realisiert. Ausgangspunkt bilden die in Abschnitt 4.4.1.4 beschriebenen Funktionen. Im Folgenden werden die Basisrevisionskontrollfunktionalitäten erläutert.

Innerhalb von R43ples kann auf Graphenebene revisioniert werden. Die jeweiligen Graphen, aber auch die Revisionsinformation, werden dabei im angeschlossenen Triple Store abgelegt, da es sich bei R43ples um einen Proxy vor einem bestehenden Triple Store handelt. Es können dabei mehrere Graphen parallel innerhalb eines Triple Stores revisioniert werden. Für die Realisierung dieser Funktionalität werden die Referenzen auf die revisionierten Graphen in einem eigenen Graphen namens *http://eatld.et.tu-dresden.de/r43ples-revisions* gespeichert. Dieser ist gleichzusetzen mit der Menge der Revisionsgraphen Γ. Jeder der beinhalteten Graphen wird mittels *rmo:RevisionGraph* beschrieben und hat eine Referenz auf einen zugehörigen Revisionsgraphen, wie in Abbildung 5.5 dargestellt. In diesem Revisionsgraphen erfolgt die semantische Beschreibung der Revisionsinformation unter Nutzung der RMO. Generell kann festgehalten werden, dass alle Mengen, die in Gleichung 4.7 Teilmengen von $\mathcal{S}$ sind, als eigene Graphen im Triple Store erzeugt werden, um die entsprechenden Tripel vorzuhalten. Alle anderen Definitionen werden mittels der semantischen Beschreibung ausgedrückt. Bei den Identifikatoren, in den gegebenen Gleichungen der Einfachheit halber mittels $\mathbb{N}^+$ angegeben, wurden innerhalb von R43ples entweder ebenfalls natürliche Zahlen für die Revisionsnummern verwendet. Des Weiteren wurden Zeichenketten für die Kennzeichnung von Entwicklungszweigen und Tags sowie generierte URIs zur Identifizierung von Ressourcen und anderen Graphen verwendet.

Initiale Erstellung und Löschung eines Revisionsgraphen Für die Erstellung eines neuen Graphen nach Gleichung 4.12 muss neben dem Graphnamen der Nutzer sowie eine Commitnachricht spezifiziert werden (Listing 1). Der neu erstellte Graph wird direkt unter Revisionskontrolle gestellt, wie beispielhaft in Abbildung B.1 dargestellt. Der *master*-Entwicklungszweig wird dabei automatisch erzeugt.

```
1 USER "bob" MESSAGE "Create a new graph"
2 CREATE GRAPH <test>
```

Listing 1: Anfrage für die Erstellung eines neuen Graphen

Ebenso ist es möglich einen bestehenden Revisionsgraphen zu löschen, wobei Gleichung 4.13 zur Anwendung kommt. Die Ausführung der Löschung wird durch die in Listing 2 dargestellte Anfrage ausgelöst und führt dazu, dass der Revisionsgraph mit aller Information rückstandslos entfernt wird.

```
1 DROP GRAPH <test>
```

Listing 2: Anfrage für die Löschung eines bestehenden Graphen

Erstellung eines Entwicklungszweiges Innerhalb des Graphen können neue Entwicklungszweige mittels der in Listing 3 angegebenen Anfrage erstellt werden. Ausgangspunkt bildet dabei, wie in Gleichung 4.14 dargestellt, eine Revision in der Historie des Graphen. Des Weiteren muss der Nutzer, eine Commitnachricht und ein Name für den neuen Entwicklungszweig angegeben werden. Ein zugehöriges Beispiel der Abbildung im Informationsraum ist in Abbildung B.2 aufgeführt.

```
1 USER "bob" MESSAGE "Create experiment branch"
2 BRANCH GRAPH <test> REVISION "2" TO "experiment"
```

Listing 3: Anfrage für die Erstellung eines neuen Entwicklungszweiges

Erstellung eines Tags Analog zu Entwicklungszweigen können neue Tags nach Gleichung 4.15 angelegt werden. Die Anfrage (Listing 4) unterscheidet sich im Wesentlichen nur durch den Austausch des Schlüsselwortes *Branch* durch *Tag* sowie die die Beschreibung im Informationsraum, wie in Abbildung B.3 beispielhaft dargestellt.

```
1 USER "bob" MESSAGE "Tag version v1.0"
2 TAG GRAPH <test> REVISION "2" TO "v1.0"
```

Listing 4: Anfrage für die Erstellung eines neuen Tags

Erstellung und Revidierung eines Commits Auf Basis eines erstellten Graphen können weitere Commits durchgeführt werden (Listing 5). Es muss wiederum Nutzer und Commitnachricht angegeben werden, um den Commit zu kennzeichnen. Auf Basis eines Entwicklungszweiges werden die Änderungen angegeben, die das Hinzufügen beziehungsweise das Löschen von Tripeln bewirken, wie in Gleichung 4.16 beschrieben. Ein visuelles Beispiel ist in Abbildung B.4 dargestellt.

```
1  USER "bob" MESSAGE "Create update"
2  INSERT {
3      GRAPH <test> BRANCH "master" {
4          <b> <c> <d> .
5      }
6  }
7  DELETE {
8      GRAPH <test> BRANCH "master" {
9          <e> <f> <g> .
10     }
11 }
```

Listing 5: Anfrage für die Erstellung eines neuen Commits

Die Revidierung eine Commits ist, wie auch in Gleichung 4.17, immer nur in Bezug auf den letzten Commit eines Entwicklungszweiges möglich. Für die Durchführung steht die in Listing 6 dargestellte Anfrage zur Verfügung. Durch diese erfolgt die entsprechende Manipulation im Informationsraum, die an einem Beispiel in Abbildung B.5 dargestellt ist.

```
1 USER "bob" MESSAGE "Revert last commit"
2 REVERT GRAPH <test> BRANCH "master"
```

Listing 6: Anfrage für die Revidierung eines Commits

Zugriff auf Revisionsinhalte Für den Zugriff auf Revisionsinhalte werden SPARQL-Anfragen verwendet, wobei jeweils durch ein zusätzliches Schlüsselwort namens *REVISION* der Revisionsstand gekennzeichnet wird, der für die Abfrage genutzt werden soll. Hierfür muss das Schlüsselwort nach einer Graphdefinition in der Anfrage folgen. Für die Identifikation des Revisionsstandes können der Revisionsidentifikator oder ein Identifikator von einem Entwicklungszweig beziehungsweise von einen Tag genutzt werden. Es ist dabei auch möglich, unterschiedliche Graphen mit unterschiedlichen Revisionen gleichzeitig in einer Anfrage abzufragen. Neben SELECT, beispielhaft in Listing 7 dargestellt, werden ebenso CONSTRUCT und ASK als Anfragearten unterstützt. Die Wiederherstellung der notwendigen Revisionsinhalte erfolgt nach den in [GHU16] vorgestellten Verfahren.

```
1 SELECT *
2 WHERE {
3   GRAPH <graph> REVISION "23" {?s ?p ?o}
4 }
5
6 SELECT *
7 WHERE {
8     GRAPH <graph> REVISION "master" {?s ?p ?o}
9 }
```

Listing 7: Anfrage für den Zugriff auf Revisionsinhalte

5.2.3 Aggregation von High-Level-Changes

Für die Aggregation von atomaren Änderungen zu High-Level-Changes wird eine Teilkomponente innerhalb des Kerns von R43ples angelegt, die die benötigten Funktionalitäten zur Verfügung stellt. Für die Ausführung der Funktionalitäten steht eine neue Anfrage zur Verfügung, die beispielhaft in Listing 8 dargestellt ist. Durch diese können Änderungen, analog zu Gleichung 4.18, zwischen zwei aufeinanderfolgenden Revisionen zu High-Level-Changes aggregiert werden.

```
1 AGG GRAPH <test> REVISION "1" TO REVISION "2"
```

Listing 8: Anfrage für die Aggregation

Die technologiespezifische Umsetzung der Gleichung 4.19 erfolgt auf Grundlage der durch Papavasileiou *et al.* [Pap+13] beschriebenen Regelsätze. Für die Integration der Regelsätze in das semantische Gesamtgefüge von R43ples wird eine neue Ontologie namens Aggregation and Evolution Rules Ontology (AERO) definiert, die sowohl die Aggregation der atomaren Änderungen als auch die darauf aufbauenden Co-Evolutionen semantisch beschreibt. Abbildung 5.6 zeigt den Ausschnitt der AERO, der die Beschreibung der Aggregation ermöglicht, inklusive einiger Erweiterungen der RMO, um die detektierten High-Level-Changes zu beschreiben.

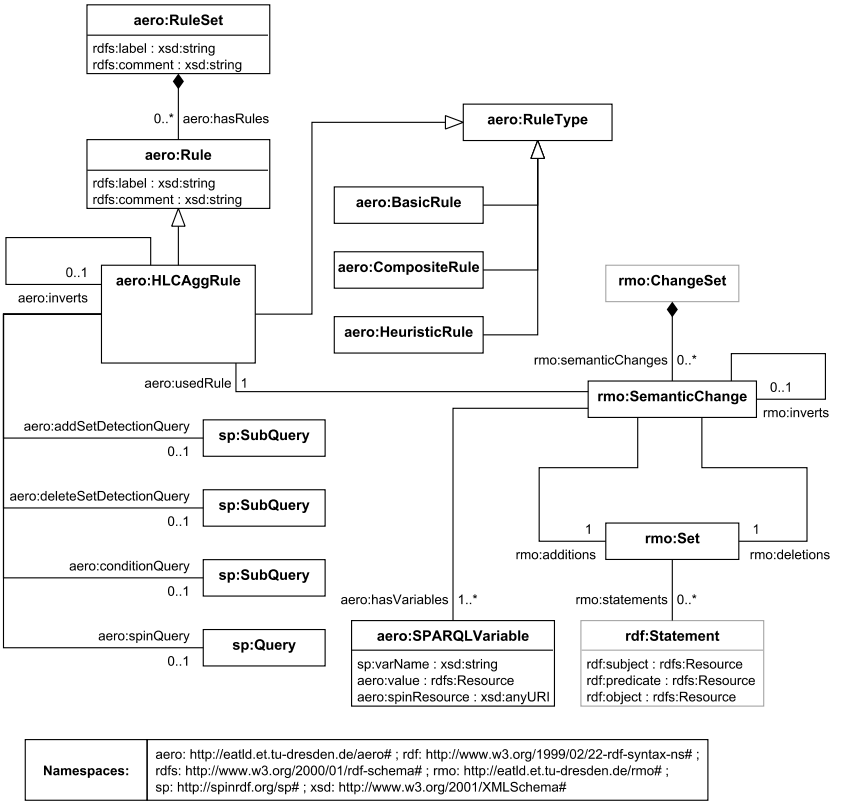


Abbildung 5.6: Ausschnitt der AERO für die Aggregation als UML-Modell inklusive zusätzlicher Erweiterungen der RMO

Grundidee für die Anwendung der Regelsätze ist die Nutzung von SPARQL-Anfragen. Mittels dieser können die Matchings in den Änderungen detektiert werden. Für die Umsetzung der Anfragen wird auf SPARQL Inferencing Notation (SPIN)[2] zurückgegriffen. Im Gegensatz zum Nachfolger namens Shapes Constraint Language (SHACL)[3] erlaubt SPIN die Referenzierung von einzelnen Elementen innerhalb einer SPARQL-Anfrage. Dies wird im Folgenden für die Modellierung als wesentlicher Bestandteil verwendet.

Die grundlegenden Elemente für die Beschreibung der detektierten High-Level-Changes wurden aus Abbildung 4.10 übernommen und in die RMO überführt. Durch die Aggregation werden einem *rmo:ChangeSet* die zugehörigen, detektierten *rmo:SemanticChange*s zugeordnet. Mittels *aero:usedRule* wird zudem eine semantische Verbindung zu der zugrunde liegenden Regel hergestellt. Alle vorhandenen Regeln werden dabei innerhalb von R43ples in einem RDF-Graphen vorgehalten und bei Bedarf von R43ples ausgewertet. Eine *aero:HLCAggRule* besitzt, wie von Papavasileiou *et al.* [Pap+13] beschrieben, ein Matching für die Menge der hinzugefügten Tripel (*aero:addSetDetectionQuery*), ein Matching für die Menge der gelöschten Tripel (*aero:deleteSetDetectionQuery*), die Möglichkeit der Spezifikation von weiteren Bedingungen (*aero:conditionQuery*) sowie einen Typen (*RuleType*). Diese einzelnen Elemente sind dabei in einer SPIN-Anfrage gekapselt (*aero:spinQuery*). Ein Beispiel für die zugehörige SPARQL-Anfrage ist im Anhang in Listing 16 dargestellt. Darin enthalten sind Platzhalter für die RDF-Graphen, die abgefragt werden müssen. Diese Platzhalter werden zur Laufzeit von R43ples entsprechend der getätigten Anfrage ersetzt. Dieser Ersetzungsprozess ist notwendig, da SPIN die R43ples-spezifischen SPARQL-Erweiterungen nicht unterstützt.

Für die Detektion der Matchings werden im ersten Schritt die verfügbaren Regelsätze durch R43ples ermittelt und dann nacheinander angewendet. Hierbei wird jeweils die gesamte SPIN-Anfrage zurück in eine SPARQL-Anfrage gewandelt, wobei die notwendigen Ersetzungen vorgenommen werden. Anschließend erfolgt die Ausführung der Anfrage auf dem Endpunkt von R43ples, der die R43ples-spezifischen SPARQL-Erweiterungen unterstützt. Das Ergebnis wird dann direkt im Revisionsgraphen zum entsprechenden *rmo:ChangeSet* gespeichert. Hierfür werden dann die Matchings der hinzugefügten und der gelöschten Tripel nochmals einzeln ausgewertet und anschließend zusammen mit den Ergebnissen der Gesamtanfrage gespeichert. Die Tripel werden dabei nicht in separaten RDF-Graphen gespeichert, sondern semantisch als *rdf:Statement*s unterteilt in Subjekt, Prädikat und Objekt abgelegt, die den *rmo:additions* beziehungsweise *rmo:deletions* zugeordnet sind. Die Ergebnisse der Gesamtabfrage werden in Form der SPARQL-Variablen gespeichert. Da jede dieser Variablen eine Repräsentation in SPIN besitzt, wird auf die entsprechende Ressource mittels *aero:spinResource* verwiesen. Durch die Bereitstellung dieser Information kann dann zum Beispiel eine Visualisierungskomponente die detektierten High-Level-Changes grafisch aufbereiten. Ein Beispiel für die im Revisionsgraphen abgelegte Information ist in Abbildung B.6 dargestellt.

[2] http://spinrdf.org/ (besucht am 29.11.2020)
[3] https://www.w3.org/TR/shacl/ (besucht am 29.11.2020)

5.2.4 Zusammenführung divergierter Entwicklungszweige

Die Zusammenführung von divergierten Entwicklungszweigen erfolgt wiederum mittels einer Teilkomponente im Kern von R43ples. Für die unterschiedlichen Methoden der Zusammenführung, wie in Abschnitt 4.4.3.1 eingeführt, stehen entsprechende neue Anfragen zur Verfügung, die über jeweilige Schlüsselwörter verfügen. Hierbei wird grundlegend in *MERGE* und *PICK* unterschieden.

3-Wege-Merges und Fast Forward Zwischen 3-Wege-Merges und Fast Forwards wird in Bezug auf die Anfrage nicht unterschieden. So kommt als Basis jeweils eine Anfrage, wie in Listing 9 dargestellt, zum Einsatz. Hierbei müssen der Nutzer und die Commitnachricht angegeben werden, um den Commit für die Zusammenführung näher zu spezifizieren. Des Weiteren werden die zusammenzuführenden Entwicklungszweige definiert und damit wird gleichzeitig festgelegt, auf welchem Entwicklungszweig die resultierende Revision liegen soll. Bei der Ausführung der Anfrage wird der Revisionsverlauf analysiert und darauf basierend entschieden, ob ein Fast Forward entsprechend Gleichung 4.24 vorgenommen werden kann oder ob ein 3-Wege-Merge, wie in Gleichung 4.20, durchgeführt werden muss. Das Ergebnis wird dann wiederum semantisch mittels der RMO beschrieben. Beispiele hierfür sind in Abbildung B.7 (3-Wege-Merge Commit) und Abbildung B.8 (Fast Forward Commit) dargestellt.

```
1  USER "bob" MESSAGE "3-Way-Merge or Fast Forward"
2  MERGE GRAPH <test> BRANCH "experiment" INTO "master"
```

Listing 9: 3-Wege-Merge Anfrage

Die Berechnung der zu speichernden Änderungen zwischen den bestehenden Revisionen und der neu erzeugten Revision, wie in Gleichung 4.21 eingeführt, erfolgt mittels dem in Abschnitt 4.4.3.2 vorgestellten Vorgehen zur Erkennung von transienten Effekten auf atomarer Änderungsebene und der Ableitung einer Konfliktlösung. Die Umsetzung erfolgt dabei in Anlehnung an [Hen14] und [HGU16]. Die in diesen Arbeiten eingeführten Ontologien werden in die Merge Management Ontology (MMO) zusammengeführt, wodurch redundante Definitionen in den Ontologien und zusätzliche Abbildungen zwischen diesen in der Implementierung vermieden werden. Das UML-Modell der MMO ist in Abbildung 5.7 dargestellt.

Die MMO ermöglicht einerseits die Abbildung einer Konfliktbeschreibungsmatrix, wie in Abbildung 4.12 dargestellt, um mögliche Konflikte und deren automatisierte Auflösung spezifisch oder allgemein für Revisionsgraphen festzulegen (*mmo:hasDefaultSDG*). Auf der anderen Seite erlaubt diese Ontologie die Beschreibung der Änderungen entlang eines Pfades (*RevisionProgress*) für die interne semantische Verarbeitung der Anwendung von Gleichung 4.31. Schließlich kann mittels der MMO die Beschreibung einer möglichen Konfliktlösung (*DifferenceGroup*s), wie in Gleichung 4.36 eingeführt, durchgeführt werden, um diese einem Client zur Verfügung stellen zu können.

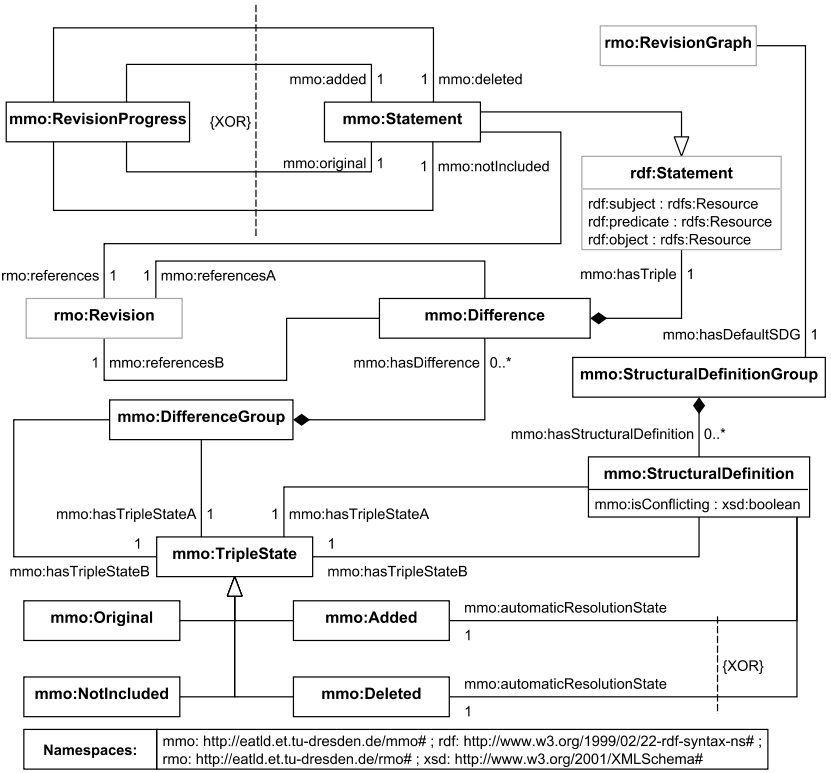


Abbildung 5.7: MMO als UML-Modell

Für die Interaktion des Clients mit dem System stehen außerdem Erweiterungen der Grundanfrage aus Listing 9 zur Verfügung. Im Fall, dass die in Listing 9 aufgeführte Anfrage aufgrund von Konflikten nicht ausgeführt werden kann, erhält der Nutzer das Modell der detektierten Konflikte entsprechend der MMO zurück und kann darauf basierend dem System selbst eine Konfliktlösung bereitstellen. Dies wird mittels der Anfrage in Listing 10 durchgeführt, da im *WITH*-Teil die entsprechenden Tripel angegeben werden können, die aus der Menge der konfliktbehafteten Tripel in der zusammengeführten Revision enthalten sein sollen.

```
1  USER "bob" MESSAGE "3–Way–Merge with a WITH part"
2  MERGE GRAPH <test> BRANCH "experiment" INTO "master" WITH {
3    <http://test.com/Carlos> <http://test.com/knows> <http://test
       .com/Danny> .
4    <http://test.com/Franz> <http://test.com/knows> <http://test.
       com/Silvia> .
5  }
```

Listing 10: 3-Wege-Merge Anfrage mit nutzerdefinierter Konfliktlösung

Im Gegensatz dazu erlaubt die Anfrage in Listing 11 die automatisierte Auflösung von Konflikten. Grundlage hierfür bilden die mittels *mmo:hasDefaultSDG* beschriebenen Konfliktlösungen innerhalb der Konfliktmatrix.

```
1  USER "bob" MESSAGE "Commit message for automatic 3–Way–Merge"
2  MERGE AUTO GRAPH <test> BRANCH "experiment" INTO "master"
```

Listing 11: 3-Wege-Merge Anfrage mit automatisierter Konfliktlösung

Schließlich besteht die Möglichkeit, unabhängig von den bereits aufgeführten Anfragen den Inhalt der neu erstellten Revision zu definieren. Dieser muss mittels der entsprechenden Tripel im *WITH*-Teil der Anfrage angegeben werden. Listing 12 stellt hierzu ein Beispiel einer solchen Anfrage dar.

```
1  USER "bob" MESSAGE "Manual 3–Way–Merge"
2  MERGE MANUAL GRAPH <test> BRANCH "experiment" INTO "master"
       WITH {
3    <http://test.com/Carlos> <http://test.com/knows> <http://test
       .com/Danny> .
4    <http://test.com/Franz> <http://test.com/knows> <http://test.
       com/Silvia> .
5  }
```

Listing 12: Manuelle 3-Wege-Merge Anfrage

Pick Die in Gleichung 4.23 beschriebene Funktionalität wird durch die in Listing 13 dargestellte Anfrage realisiert. Dabei muss der Nutzer und eine Commitnachricht angegeben werden. Das Pick wird für den angegebenen Bereich der Revisionen ausgeführt, wobei Start- und Endrevision eingeschlossen sind. Die ausgewählten Änderungen werden anschließend auf den spezifizierten Entwicklungszweig angewendet. Im Fall, dass nur die Änderungen einer Revision wiederverwendet werden sollen, kann die in Listing 14 dargestellte Anfrage genutzt werden. Ein Beispiel für die semantische Beschreibung eines Pick Commits ist in Abbildung B.9 dargestellt.

```
1 USER "bob" MESSAGE "Pick"
2 PICK GRAPH <test> REVISION "2" TO REVISION "4" INTO BRANCH "
      master"
```

Listing 13: Pick Anfrage

```
1 USER "bob" MESSAGE "Simple pick"
2 PICK GRAPH <test> REVISION "2" INTO BRANCH "master"
```

Listing 14: Vereinfachte Pick Anfrage

5.3 Evolutionsmechanismen

Auf Basis von aggregierten Änderungen können durch eine weitere Teilkomponente von R43ples Co-Evolutionen durchgeführt werden. Die Ausführung der zugehörigen Funktionalität, wie in Gleichung 4.43 definiert, erfolgt durch eine neue Anfrage, die beispielhaft in Listing 15 dargestellt ist. Detektierte High-Level-Changes zwischen zwei Revisionen in einem Revisionsgraphen können durch diese Anfrage an abhängige Revisionsgraphen propagiert werden. In der vorliegenden Implementierung müssen die Revisionen direkt aufeinander folgen. Des Weiteren werden keine expliziten Verbindungen innerhalb eines separaten Graphen ausgewertet, sondern es werden nur Co-Evolutionen durchgeführt, deren Abhängigkeit direkt aus den Inhalten heraus besteht.

```
1 USER "bob" MESSAGE "Coevolve"
2 COEVO GRAPH <test> REVISION "1" TO REVISION "2"
```

Listing 15: Anfrage für die Co-Evolution

Für die High-Level-Changes wird die Implementierung aus Abschnitt 5.2.3 als Grundlage verwendet. Hierbei kann auch auf die bereits eingeführten Ontologien zurückgegriffen werden, die jedoch um die Co-Evolutionsaspekte erweitert werden müssen. Die notwendigen Erweiterungen innerhalb der RMO und der AERO sind in Abbildung 5.8 dargestellt.

Ebenso wie in Abschnitt 5.2.3 wird bei der Beschreibung der Regelsätze auf SPIN zurückgegriffen. Die Regeln (*aero:CoEvoRule*) sind dabei von *aero:HLCAggRule* abgeleitet und erweitern diese um den Co-Evolutionspart. Diese Erweiterung besteht aus einer SPARQL-SELECT-Anfrage, mittels dieser identifiziert werden kann, ob eine Abhängigkeit zu einem zu prüfenden Modell vorliegt. Die eigentliche Co-Evolution wird dann mittels der spezifizierten Änderungen in Bezug auf die zu tätigenden Hinzufügungen (*aero:addSetInsertQuery*) und die zu tätigenden Löschungen (*aero:deleteSetInsertQuery*) durchgeführt. Hierbei handelt es sich im SPARQL-INSERT-Anfragen, die innerhalb von SPIN mittels *sp:Modify* abgebildet werden. Als Variablen in den SPARQL-Anfragen werden die gleichen verwendet wie bei der Aggregation zu High-Level-Changes, wodurch direkt auf die Ergebnisse dieser Anfragen bei der Co-Evolution zurückgegriffen werden kann. Im Anhang in Listing 17 sind beispielhaft die SPARQL-Anfragen dargestellt, die

für die Regelbeschreibung verwendet werden. Dieses Beispiel erweitert die bereits vorgestellte Beispielanfrage aus Listing 16 um den Co-Evolutionsteil. Die darin enthaltenen Platzhalter werden zur Laufzeit von R43ples mit den konkreten Werten ersetzt.

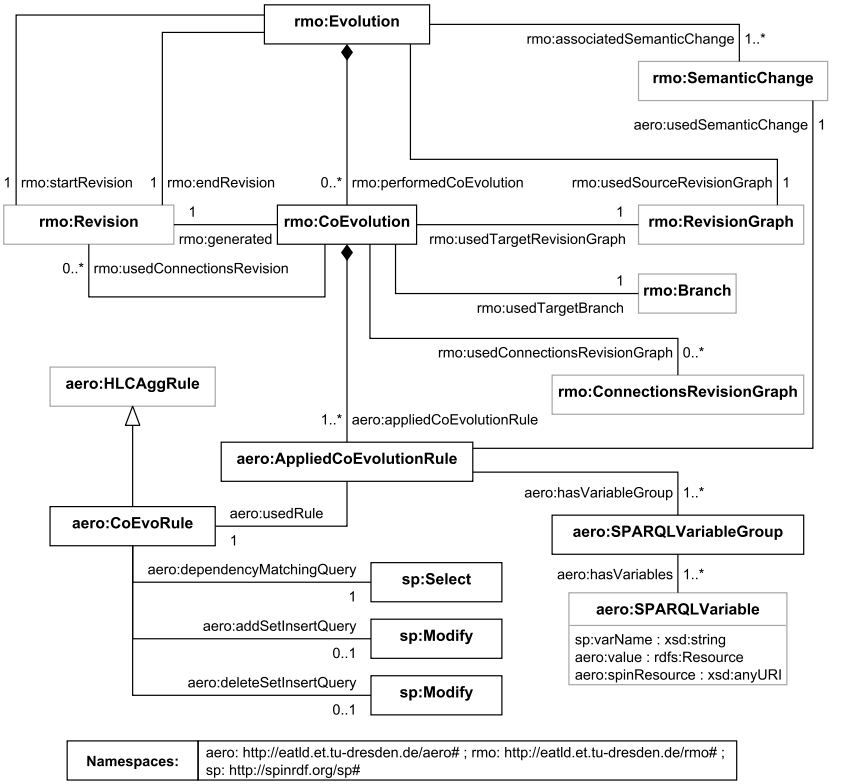


Abbildung 5.8: Erweiterung der AERO für die Co-Evolution als UML-Modell inklusive zusätzlicher Erweiterungen der RMO

Die Elemente zur Beschreibung der Co-Evolution wurden aus Abbildung 4.17 übernommen und in die RMO überführt. Die Verknüpfung der *rmo:CoEvolution* mit den zugrunde liegenden Regeln erfolgt mittels *aero:appliedCoEvolutionRule* und erlaubt nachzuvollziehen, welche Regeln für die Co-Evolution verwendet wurden. Des Weiteren werden durch das referenzierte Element *aero:AppliedCoEvolutionRule* die SPARQL-Variablen inklusive deren Werte beschrieben, auf deren Basis die zu erzeugenden oder zu löschenden Tripel in den abhängigen Modellen definiert werden. Da innerhalb eines Regelsatzes mehrere Variablen verwendet werden können, aber auch potenziell mehrfache Matchings in den

abhängigen Modellen existieren können, werden die Variablen pro Matching mittels *aero:SPARQLVariableGroup* gruppiert. Überdies wird der *aero:AppliedCoEvolutionRule* ein *rmo:SemanticChange* mittels *aero:usedSemanticChange* zugeordnet, um den Auslöser der Co-Evolution nachvollziehbar zu beschreiben.

Die Durchführung der Co-Evolutionen basiert auf dem in Gleichung 4.43 dargestellten Verfahren. Dabei wird im ersten Schritt analysiert, ob bereits eine Aggregation der atomaren Änderungen vorgenommen wurde. Für den Fall, dass diese Aggregation noch nicht vorliegt, wird diese aus der Co-Evolution heraus angestoßen. Im Anschluss werden auf Basis der bereits vorliegenden oder der neu durchgeführten Aggregation alle Revisionsgraphen auf deren Abhängigkeit geprüft. Hierzu werden die Matchinganfragen der anzuwendenden Regeln ausgeführt. Bei der Detektion von Matchings werden die durchzuführenden Änderungen berechnet und aus diesen ein neuer Commit innerhalb des abhängigen Revisionsgraphen erzeugt. Das Ergebnis der Co-Evolution wird in der derzeitigen Implementierung immer nur für den *master*-Zweig durchgeführt. Nachdem alle Co-Evolutionen durchgeführt wurden, wird die zugehörige semantische Beschreibung innerhalb des Revisionsgraphen der Co-Evolution als neuer Commit gespeichert. Ein Beispiel für die durch den Commit beschriebene Information ist in Abbildung B.10 dargestellt.

5.4 Weitere Arbeiten in diesem Bereich

In weiteren Arbeiten wurden bereits erste Visualisierungskonzepte für die Interaktion mit R43ples entwickelt. Hierbei wurde auch teilweise auf niedrig priorisierte Anforderungen an ein RMS eingegangen. Im Bereich der Zusammenführung von divergierten Entwicklungszweigen wird in [Yan15] ein Ansatz für die Umsetzung von Mergeprozessen vorgestellt. Bei diesem Ansatz wird der Nutzer schrittweise durch den Mergevorgang geführt und erhält eine visuelle Darstellung von aufgetretenen strukturellen Konflikten, die dann visuell gelöst werden können (Bezug zu Anforderungen A-101, A-202). Für die Co-Evolution wird in [Fun17] ebenfalls ein geführter Prozess dargelegt, der die Visualisierung von Abhängigkeiten innerhalb eine RMS ermöglicht und der Nutzer im Anschluss die durchzuführenden Co-Evolutionen konfigurieren kann (Bezug zu Anforderungen A-101, A-301, A-302). Zum Beispiel kann der Nutzer auswählen welche der abhängigen Modelle co-evolviert werden sollen.

Ebenfalls existieren erste Ansätze für die Integration einer Benutzerverwaltung in R43ples, wie in [Pha16] vorgestellt (Bezug zu Anforderungen A-102, A-103). Dabei wird als Technologie für die Benutzerverwaltung auf Lightweight Directory Access Protocol (LDAP) zurückgegriffen. In einer weiteren Arbeit wird ein Ansatz vorgestellt, um die Integration und Synchronisation von OPC UA und Semantic Web Informationsmodellen zu ermöglichen [Ahr18]. Durch diesen Ansatz können auch Änderungen innerhalb von OPC UA mittels R43ples semantisch revisioniert werden, wodurch erste Rückschlüsse auf die Übertragbarkeit der technologieunabhängigen Beschreibungen eines RMS gezogen werden können.

6 Verifikation

Mittels der Verifikation wird im Folgenden der Nachweis erbracht, dass das konzeptionierte und implementierte RMS funktionsfähig ist. Hierfür werden jeweils unterschiedliche Strategien für den Nachweis angewendet. In Bezug auf die konzeptionierten formalen mathematischen Definitionen ist eine vollständige Beweisführung für deren Korrektheit nur bedingt möglich, da die zu beweisenden Sachverhalte überwiegend direkt in den Definitionen als Randbedingungen angegeben sind und die notwendigen Beweise daher trivial wären. Zur Sicherstellung, dass die mathematischen Definitionen auch in Kombination anwendbar sind, wird im ersten Schritt deren Funktionsweise an einem Beispiel, das die wesentlichen Funktionen abdeckt, nachvollzogen. Im Anschluss erfolgt die Beweisführung, dass innerhalb eines Revisionsgraphen durch den Nutzer immer ein beliebiger Revisionsinhalt erzeugt werden kann und der Nutzer dementsprechend nicht durch die mathematischen Definitionen und deren Randbedingungen in der Erstellung und Änderung von Revisionsinhalten eingeschränkt ist. Schließlich werden die allgemeinen Definitionen für die Beschreibung von verbindungsorientierten Modellen auf die Anwendung der Co-Simulation übertragen und somit ebenfalls an einem Beispiel verifiziert, dass die getätigten Definitionen anwendbar sind. Durch die Verifikation der grundlegenden Funktionalitäten wird sichergestellt, dass die Implementierung auf einer funktionsfähigen Spezifikation aufbaut. Die anschließende Verifikation der Software anhand von verschiedenen Testfällen stellt die korrekte Funktionsweise der Implementierung sicher. Hierbei werden neben den grundlegenden Funktionsweisen auch erweiterte Funktionen, wie die Co-Evolution auf Typenebene überprüft. Dabei wird einerseits gezeigt, dass die beschriebenen mathematischen Definitionen implementierbar sind und andererseits wird ebenfalls die semantische Beschreibung innerhalb der Implementierung verwendet, um die Funktionen umsetzen zu können, aber auch durch Testfälle die korrekte Erzeugung der semantischen Beschreibung sichergestellt.

6.1 Beispielhafte Nutzung der formalen Beschreibung

Anhand eines Beispiels werden im Folgenden die formalen Definitionen der Basisrevisionskontrollfunktionalitäten angewendet und schrittweise ein Revisionsgraph aufgebaut. Als Grundlage dient hierfür die im Kapitel 5 zugrunde gelegte Definition der Menge $\mathcal{S} := \mathcal{U} \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L})$. Zur vereinfachten Darstellung der konkreten Elemente wird angenommen, dass $\mathcal{U} = \mathcal{L} = \{a,b,c,d,e,f,g,h,i\}$ gilt.

Die Revisionsgraphen werden in der Menge Γ vorgehalten. Auf deren Basis kann im ersten Schritt ein neuer Revisionsgraph erzeugt werden, wie in Gleichung 6.1 dargestellt. In dem Beispiel wird ein neues Tripel *abc* (entspricht dem Statement (a,b,c)) direkt bei der Erstellung angegeben. Der resultierende Revisionsgraph ist visuell in Abbildung 6.1 dargestellt. Die *Revision 0* ist dabei die Revision, die automatisiert erstellt wird.

$$\begin{aligned}\Gamma = \boldsymbol{create}_{\Gamma}(\{abc\}) &= \emptyset \cup \{(\{1\}, (0,1,\{abc\},\emptyset), (\{1\},1,\{abc\},1), \emptyset, 1)\} \\ &= \{(\{1\}, (0,1,\{abc\},\emptyset), (\{1\},1,\{abc\},1), \emptyset, 1)\} \\ &= \{\mathcal{G}\}\end{aligned} \tag{6.1}$$

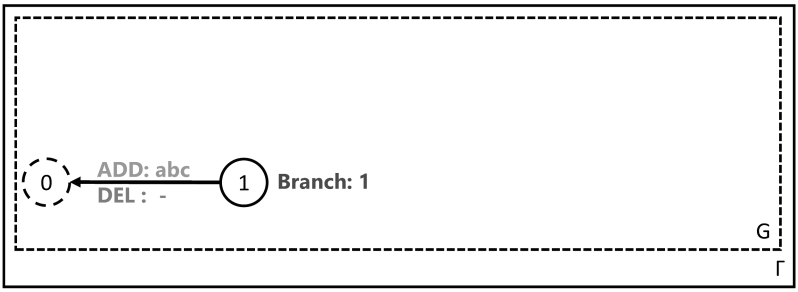


Abbildung 6.1: Revisiongraph nach der initialen Erstellung

Bei den folgenden Gleichungen wird nur die Aktualisierung von $\mathcal{G}$ dargestellt und aus Gründen der Vereinfachung auf die Darstellung von Γ verzichtet, da diese Menge nur $\mathcal{G}$ beinhaltet. Im nächsten Schritt wird ein neuer Commit auf den bestehenden *Branch 1* vorgenommen. Dieser fügt ein weiteres Tripel *ghi* hinzu und löscht das bestehende Tripel *abc*. Die Aktualisierung des Revisionsgraphen erfolgt mittels der in 4.16 definierten Funktion, die das in Gleichung 6.2 dargestellte Ergebnis erzielt. Dieses ist visuell in Abbildung 6.2 aufbereitet.

$$
\begin{aligned}
\mathcal{G} &= (R_g,C_g,B_g,T_g,n_g)\\
R_g &= \{1\}\\
C_g &= \{(0,1,\{abc\},\emptyset)\}\\
B_g &= \{(\{1\},1,\{abc\},1)\}\\
T_g &= \emptyset\\
n_g &= 1
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{G}' &= \boldsymbol{commit_{\mathcal{G}}}\,(1,\{ghi\},\{abc\}) = (R'_g,C'_g,B'_g,T_g,n_g)\\
C^{+} &= \{ghi\}\\
C^{-} &= \{abc\}\\
n_b &= 1
\end{aligned}
\tag{6.2}
$$

Anwendung der Gleichung 4.16 ergibt:

$$
\begin{aligned}
R'_g &= \{1,2\}\\
C'_g &= \{(0,1,\{abc\},\emptyset),(1,2,\{ghi\},\{abc\})\}\\
B'_g &= \{(\{1,2\},2,\{ghi\},1)\}\\
T_g &= \emptyset\\
n_g &= 1
\end{aligned}
$$

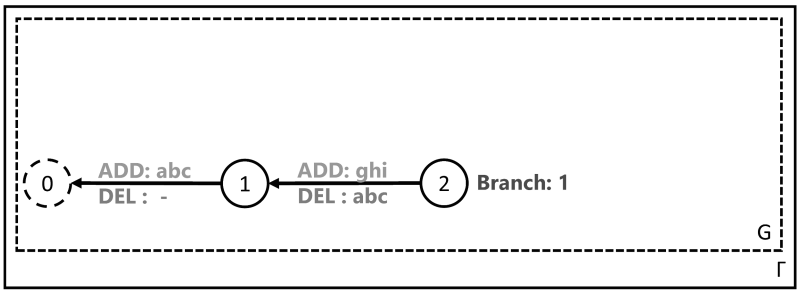


Abbildung 6.2: Revisiongraph nach dem ersten Commit

Im Anschluss erfolgt die Erstellung eines neuen Entwicklungszweiges. Ausgangspunkt bildet die *Revision 1*. Hierzu wird Gleichung 4.14 angewendet, die einen neuen *Branch 2* erzeugt. Die Anwendung der Funktion ist in Gleichung 6.3 dargelegt und Abbildung 6.3 zeigt die schematische Darstellung des Revisionsgraphen. In dieser Darstellung ist der neue Entwicklungszweig vorerst nur als neue Referenz auf *Revision 1* dargestellt

$$
\begin{aligned}
\mathcal{G} &= (R_g,C_g,B_g,T_g,n_g)\\
R_g &= \{1,2\}\\
C_g &= \{(0,1,\{abc\},\emptyset),(1,2,\{ghi\},\{abc\})\}\\
B_g &= \{(\{1,2\},2,\{ghi\},1)\}\\
T_g &= \emptyset\\
n_g &= 1
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{G}' &= \boldsymbol{branch_{\mathcal{G}}}\,(1) = (R_g,C_g,B'_g,T_g,n_g) \qquad (6.3)\\
r_x &= 1
\end{aligned}
$$

Anwendung der Gleichung 4.14 ergibt:

$$
\begin{aligned}
R_g &= \{1,2\}\\
C_g &= \{(0,1,\{abc\},\emptyset),(1,2,\{ghi\},\{abc\})\}\\
B'_g &= \{(\{1,2\},2,\{ghi\},1),(\{1\},1,\{abc\},2)\}\\
T_g &= \emptyset\\
n_g &= 1
\end{aligned}
$$

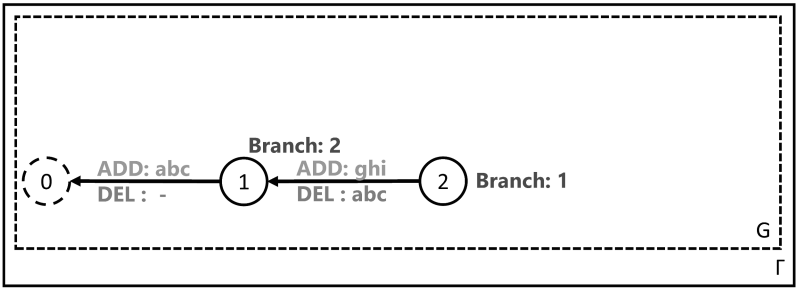


Abbildung 6.3: Revisiongraph nach der Erstellung eines neuen Entwicklungszweiges

Nachdem der neue Entwicklungszweig erzeugt ist, wird ein neuer Commit auf diesen Entwicklungszweig vorgenommen, der ein neues Tripel def hinzufügt. Die Anwendung der Gleichung 4.16 erfolgt in Gleichung 6.4 und die zugehörige Visualisierung in Abbildung 6.4.

$$
\begin{aligned}
\mathcal{G} &= (R_g,C_g,B_g,T_g,n_g)\\
R_g &= \{1,2\}\\
C_g &= \{(0,1,\{abc\},\emptyset),(1,2,\{ghi\},\{abc\})\}\\
B_g &= \{(\{1,2\},2,\{ghi\},1),(\{1\},1,\{abc\},2)\}\\
T_g &= \emptyset\\
n_g &= 1
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{G}' &= \boldsymbol{commit_{\mathcal{G}}}\,(2,\{def\},\emptyset) = (R'_g,C'_g,B'_g,T_g,n_g)\\
C^{+} &= \{def\}\\
C^{-} &= \emptyset\\
n_b &= 2
\end{aligned}
\tag{6.4}
$$

Anwendung der Gleichung 4.16 ergibt:

$$
\begin{aligned}
R'_g &= \{1,2,3\}\\
C'_g &= \{(0,1,\{abc\},\emptyset),(1,2,\{ghi\},\{abc\}),(1,3,\{def\},\emptyset)\}\\
B'_g &= \{(\{1,2\},2,\{ghi\},1),(\{1,3\},3,\{abc,def\},2)\}\\
T_g &= \emptyset\\
n_g &= 1
\end{aligned}
$$

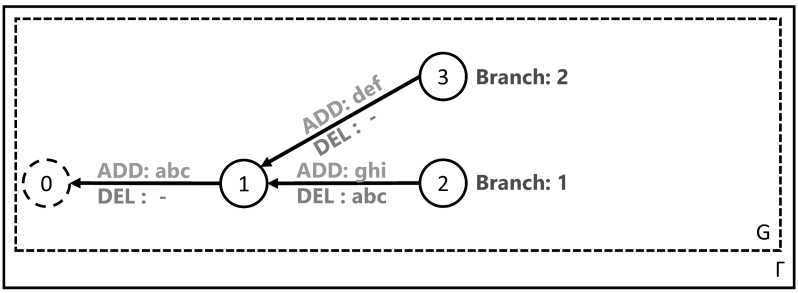


Abbildung 6.4: Revisiongraph nach dem Commit auf dem neu erstellten Entwicklungszweig

Die *Revision 1* wird nachfolgend mit einem Tag versehen. Hierzu erfolgt die Anwendung der Gleichung 4.15, wie in Gleichung 6.5 beschrieben. Der resultierende Revisionsgraph wird in Abbildung 6.5 gezeigt.

$$
\begin{aligned}
\mathcal{G} &= (R_g,C_g,B_g,T_g,n_g)\\
R_g &= \{1,2,3\}\\
C_g &= \{(0,1,\{abc\},\emptyset),(1,2,\{ghi\},\{abc\}),(1,3,\{def\},\emptyset)\}\\
B_g &= \{(\{1,2\},2,\{ghi\},1),(\{1,3\},3,\{abc,def\},2)\}\\
T_g &= \emptyset\\
n_g &= 1
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{G}' &= \boldsymbol{tag}_{\mathcal{G}}\,(1) = (R_g,C_g,B_g,T'_g,n_g) \qquad (6.5)\\
r_x &= 1
\end{aligned}
$$

Anwendung der Gleichung 4.15 ergibt:

$$
\begin{aligned}
R_g &= \{1,2,3\}\\
C_g &= \{(0,1,\{abc\},\emptyset),(1,2,\{ghi\},\{abc\}),(1,3,\{def\},\emptyset)\}\\
B_g &= \{(\{1,2\},2,\{ghi\},1),(\{1,3\},3,\{abc,def\},2)\}\\
T'_g &= \{(1,\{abc\},1)\}\\
n_g &= 1
\end{aligned}
$$

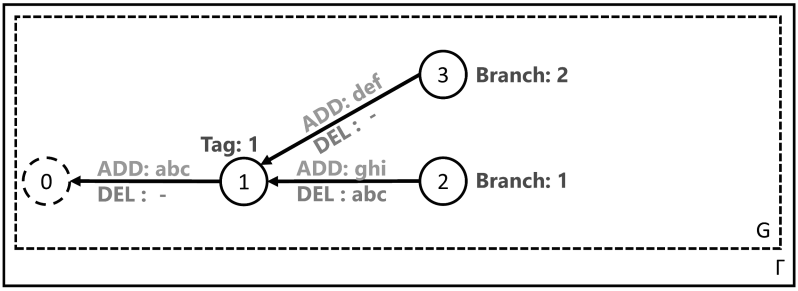


Abbildung 6.5: Revisiongraph nach der Erstellung eines neuen Tags

Im letzten Schritt wird der letzte Commit auf *Branch 1* rückgängig gemacht. Dies erfolgt durch die Anwendung der Gleichung 4.17. Im resultierenden Ergebnis, beschrieben in Gleichung 6.6, sind die Änderungen durch die Vertauschung von Hinzufügungen und Löschungen rückgängig gemacht worden. Eine visuelle Aufbereitung der durchgeführten Änderungen am Revisionsgraphen ist in Abbildung 6.6 zu finden.

$$
\begin{aligned}
\mathcal{G} &= (R_g,C_g,B_g,T_g,n_g)\\
R_g &= \{1,2,3\}\\
C_g &= \{(0,1,\{abc\},\emptyset),(1,2,\{ghi\},\{abc\}),(1,3,\{def\},\emptyset)\}\\
B_g &= \{(\{1,2\},2,\{ghi\},1),(\{1,3\},3,\{abc,def\},2)\}\\
T_g &= \{(1,\{abc\},1)\}\\
n_g &= 1
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{G}' &= \boldsymbol{revert}_{\mathcal{G}}\,(1) = (R'_g,C'_g,B'_g,T_g,n_g) \qquad (6.6)\\
n_b &= 1
\end{aligned}
$$

Anwendung der Gleichung 4.17 ergibt:

$$
\begin{aligned}
R'_g &= \{1,2,3,4\}\\
C'_g &= \{(0,1,\{abc\},\emptyset),(1,2,\{ghi\},\{abc\}),(1,3,\{def\},\emptyset),(2,4,\{abc\},\{ghi\})\}\\
B'_g &= \{(\{1,2,3\},3,\{abc\},1),(\{1,3\},3,\{abc,def\},2)\}\\
T_g &= \{(1,\{abc\},1)\}\\
n_g &= 1
\end{aligned}
$$

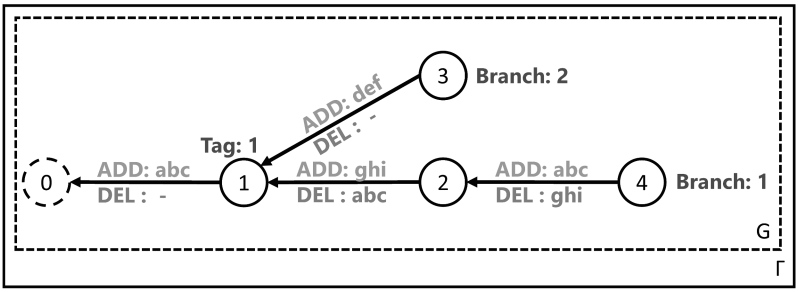


Abbildung 6.6: Revisiongraph nach der Revidierung des letzten Commits auf *Branch 1*

6.2 Nachweis der Erzeugung von beliebigen Revisionsinhalten

Ein mit dem System interagierender Nutzer muss die Möglichkeit besitzen, einen beliebigen gewünschten Revisionsinhalt durch die aufeinander aufbauende Anwendung von Commits zu erreichen. Nachfolgen wird dies formal nachgewiesen.

Ausgangspunkt ist ein beliebiger Revisionsgraph $\mathcal{G} = (R_g,C_g,B_g,T_g,n_g)$. Dieser beinhaltet bereits eine beliebige Struktur, auf deren Basis ein gewünschter Revisionsinhalt erreicht werden soll. Aus der Menge B_g wird exemplarisch ein Entwicklungszweig für den

Nachweis herangezogen. Für diesen Entwicklungszweig $b_i \in B_g$ mit $b_i = (R_{b_i}, r_{l_i}, \Upsilon_{l_i}, n_{b_i})$ wird nachgewiesen, dass mittels eines Commits das Υ_{l_i}, das den vollständigen Revisionsinhalt des Blattes des Entwicklungszweiges kennzeichnet, in ein beliebiges Υ_{l_j} überführt werden kann. Der Ansatz für die Erreichung dieses Ziels besteht darin, dass es immer möglich ist, den letzten vollständigen Inhalt als Ganzes zu löschen und durch den gewünschten neuen Inhalt zu ersetzen. Gleichung 6.7 gibt die hierfür notwendigen Definitionen an. Hieraus ergibt sich die in Gleichung 6.8 dargestellte Annahme, dass durch die Anwendung dieses Commits ein beliebiger Revisionsinhalt erzeugt werden kann.

$$\begin{aligned} \mathcal{G}' &= \boldsymbol{commit}_{\mathcal{G}} \left(n_b, C^+, C^- \right) \\ C^+ &= \Upsilon_{l_j} \\ C^- &= \Upsilon_{l_i} \\ n_b &= n_{b_i} \end{aligned} \tag{6.7}$$

$$\mathcal{G}' = \boldsymbol{commit}_{\mathcal{G}} \left(n_{b_i}, \Upsilon_{l_j}, \Upsilon_{l_i} \right) \ \forall v_i : \ v_i \in \Upsilon_{l_i}; \ \forall v_j : \ v_j \in \Upsilon_{l_j} \tag{6.8}$$

Die in Gleichung 4.16 beschriebene Aktualisierung des vollständigen Revsionsinhalts eines Entwicklungszweiges ist extrahiert in Gleichung 6.9 mittels Υ'_l dargestellt. In diesem Fall muss vor der Ausführung Gleichung 4.11 auf die Hinzufügungen und Löschungen ausgeführt werden, damit der korrekte Aufbau von C_g sichergestellt ist. Die Anwendung ist in Gleichung 6.10 dargestellt.

$$\Upsilon'_l = (\Upsilon_l \cup C^+_{stripped}) \backslash C^-_{stripped} \tag{6.9}$$

$$\begin{aligned} C^+_{stripped} &= C^+ \backslash \Upsilon_x \\ C^-_{stripped} &= C^- \cap \Upsilon_x \end{aligned}$$

$$\text{Mit } \Upsilon_x = \Upsilon_{l_i} \text{ , } C^+ = \Upsilon_{l_j} \text{ und } C^- = \Upsilon_{l_i} \text{ ergibt sich:} \tag{6.10}$$

$$\begin{aligned} C^+_{stripped} &= \Upsilon_{l_j} \backslash \Upsilon_{l_i} \\ C^-_{stripped} &= \Upsilon_{l_i} \cap \Upsilon_{l_i} \\ &= \Upsilon_{l_i} \end{aligned}$$

Die Ergebnisse aus Gleichung 6.10 können jetzt in Gleichung 6.9 mit $\Upsilon_l = \Upsilon_{l_i}$ eingesetzt werden. Hieraus ergibt sich, wie in Gleichung 6.11 dargestellt, dass $\Upsilon'_l = \Upsilon_{l_j}$. Das bedeutet, dass durch die Anwendung des in Gleichung 6.8 aufgeführten Commits immer ein gewünschter Inhalt auf einem Entwicklungszweig erzeugt werden kann, der in Bezug auf den Inhalt vollständig unabhängig von den vorangehenden Änderungen sein

kann. Für die Erreichung können beliebig viele vorangegangene Commits durchgeführt worden sein.

$$\begin{aligned}\Upsilon'_l &= (\Upsilon_{l_i} \cup (\Upsilon_{l_j} \backslash \Upsilon_{l_i})) \backslash \Upsilon_{l_i} \quad | \text{ Anwendung De Morgansche Gesetze} \\ &= (\Upsilon_{l_i} \cup \Upsilon_{l_j}) \backslash \Upsilon_{l_i} \\ &= \Upsilon_{l_j}\end{aligned} \tag{6.11}$$

6.3 Abbildung verbindungsorientierter Modelle am Beispiel der Co-Simulation

Ein Beispiel für ein verbindungsorientiertes Modell ist, wie bereits in Abschnitt 3.1.3.1 eingeführt, die Co-Simulation. In Abbildung 6.7 ist ein schematisches Beispiel einer Co-Simulation aufgeführt. Die Co-Simulation ist als Ganzes durch die Komponente *c* dargestellt. Darin enthalten sind zwei Komponenten *b1* und *b2*, die wiederum aus Teilkomponenten *a1* bis *a4* aufgebaut sind. Der Typ der jeweiligen Komponenten ist mittels <*A*>, <*B*> beziehungsweise <*C*> angegeben. Die Ports der Komponenten sind durch *I* beziehungsweise *O* gekennzeichnet. Dabei gilt, dass Ports von Teilkomponenten auch in der übergeordneten Ebene zur Verfügung stehen. Ein Beispiel hierfür ist die Verbindung zwischen den Ports *b1.O2* und *b2.I1*.

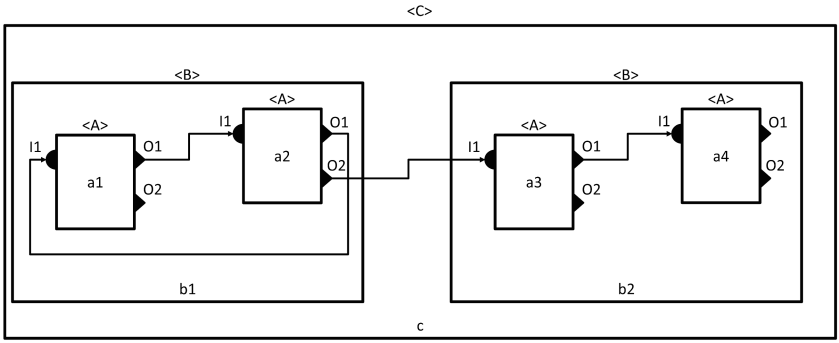


Abbildung 6.7: Beispiel eines Blockdiagramms einer Co-Simulation

Das in Abbildung 6.7 aufgeführte Beispiel lässt sich in einen Compound Graph, wie in Abschnitt 4.3 beschrieben, überführen. Die sich daraus ergebenden Strukturen von $\tilde{G}'$ und $\tilde{T}'$ sind in Abbildung 6.8 dargestellt. Das Wurzelelement bildet dabei der *Graph c*, der die übergeordnete Komponente der Co-Simulation beschreibt. Die darauf basierenden Zweige stellen dann wiederum die Subkomponenten dar und die Blätter des Baums kennzeichnen die Ports. Die Verbindungen zwischen den Ports sind nur auf der Ebene der Blätter beschrieben. Ports von übergeordneten Komponenten können, wie bereits beschrieben, aus diesen unterlagerten Ports abgeleitet werden.

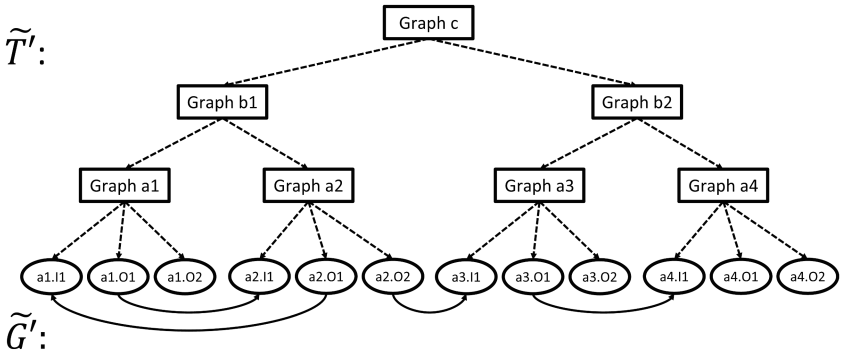


Abbildung 6.8: Abbildung des Blockdiagramms aus Abbildung 6.7 auf einen Compound Graph (gestrichelte Linien: Hierarchie-/Inklusionskanten, durchgezogene Linien: Adjazenzkanten)

Die Definitionen aus Abschnitt 4.3.1 können für den Anwendungsfall der Co-Simulation und damit allgemeiner für hierarchische verbindungsorientierte Modelle spezialisiert werden. Bei diesen Modellen können die Komponenten hierarchisch strukturiert sein und jede Komponente kann wiederum Ports anbieten. Die Relationen zwischen den Komponenten werden mittels Verbindungen zwischen den angebotenen Ports beschrieben. Die Ports sind dabei zumeist in Ein- und Ausgangsports untergliedert. Diese Untergliederung kann formal auch für die Menge der Basisknoten $\tilde{B}$ vorgenommen werden. Dies ist in Gleichung 6.12 dargestellt. Dabei beinhaltet $\tilde{I}$ die Eingangsports und $\tilde{O}$ die Ausgangsports.

$$\tilde{B} = \tilde{I} \cup \tilde{O} \tag{6.12}$$

Für die Beschreibung der Port-Port-Relationen werden die Adjazenzkanten verwendet. Da verbindungsorientierte Modelle im Allgemeinen einige Randbedingungen in Bezug auf die möglichen zu verbindenden Ports besitzen, können diese Randbedingungen analog zu den folgenden Gleichungen definiert werden. Gleichung 6.13 schränkt dabei die Konnektivität insoweit ein, dass nur Verbindungen von Ausgangs- zu Eingangsports möglich sind. Daraus folgt, dass $\tilde{G}'$ ein bipatiter Graph ist. Durch diese Einschränkung wiederum ergibt sich, dass die resultierende Klasse des Graphen eine Subkategorie von einem Compound Graph ist. Da in diesem nur Blätter mittels Adjazenzkanten verbunden sein dürfen, wird dieser auch als Clustered Graph bezeichnet [Fuh12].

$$\forall(v,w) \in E_{G'} : v \in O \wedge w \in I \tag{6.13}$$

Weitere Bedingungen zwischen Knoten und Kanten in Bezug auf die Multiplizität der möglichen Verbindungen können durch den Eingangsgrad $d^-_{\tilde{G}'}(\tilde{v})$ und den Ausgangsgrad

$d^+_{\tilde{G}'}(\tilde{v})$ eines Knotens $\tilde{v}$ beschrieben werden. Gleichung 6.14 restriktiert, dass jeder Ausgangsport nur ausgehende Verbindungen besitzt und Gleichung 6.15 schränkt ein, dass jeder Eingangsport nur eine oder keine eingehende Verbindung besitzen darf.

$$\forall \tilde{v} \in \tilde{O} : (d^-_{\tilde{G}'}(\tilde{v}) = 0) \wedge (d^+_{\tilde{G}'}(\tilde{v}) \geq 0) \tag{6.14}$$

$$\forall \tilde{v} \in \tilde{I} : (0 \leq d^-_{\tilde{G}'}(\tilde{v}) \leq 1) \wedge (d^+_{\tilde{G}'}(\tilde{v}) = 0) \tag{6.15}$$

In einigen Fällen ist es weiterhin notwendig, Abhängigkeiten zwischen Eingangs- und Ausgangsports innerhalb einer Komponente zu beschreiben. Hierfür kann die Menge der Adjazenzkanten $\tilde{E}_{\tilde{G}'}$ in zwei Mengen unterteilt werden, wie in Gleichung 6.16 beschrieben. Im Falle einer solchen weiteren Untergliederung müssen dann wiederum die Einschränkungen entsprechend erweitert werden.

$$\tilde{E}_{\tilde{G}'} = \tilde{E}_{\tilde{G}'_{connection}} \cup \tilde{E}_{\tilde{G}'_{dependency}} \tag{6.16}$$

Für die semantische Beschreibung der Co-Simulation wird im Folgenden die Beschreibung der Compound Graphs aus Abbildung 4.8 domänenspezifisch spezialisiert. Die Spezialisierung basiert auf dem Modell von Van Acker *et al.* [Van+15]. Dabei wurden die FMI-spezifischen Elemente durch generische Beschreibungen ersetzt sowie das Konzept der Bondgraphen, wie in Abschnitt 3.1.3.1 vorgestellt, integriert. Das Ergebnis ist in Abbildung 6.9 dargestellt. Von dem Blatt abgeleitet existiert der *Port* mit seinen Ableitungen *Sink* (Senke, Eingangsport) und *Source* (Quelle, Ausgangsport). Bei den Komponenten (*Component*) handelt es sich um Spezialisierungen von *Subgraph*, denen des Weiteren die *InternalDependency*s zugeordnet sind, die die internen Abhängigkeiten beschreiben und von *ConnectivityRelation* abgeleitet sind. Weiterhin existieren Signale (*Signal*), die ebenfalls eine Spezialisierung von *ConnectivityRelation* sind. Diese können überdies zu einem *Bond* gruppiert werden, der aus einem *flow*- und einem *energy*-Signal besteht. Die *CoSimulation*, abgeleitet von *CompoundGraph*, aggregiert neben den bestehenden Definitionen auch die Bonds. Die Einschränkungen, wie zum Beispiel, dass Signale eine Quelle und eine Senke miteinander verbinden, sind in der Abbildung nicht dargestellt und ergeben sich aus den vorangegangenen mathematischen Definitionen.

Auf Basis der bereitgestellten Beschreibung der Verbindungen können wiederum domänenspezifische Regelsätze für die Co-Evolution umgesetzt werden. Durch die Ableitung von dem in Abschnitt 4.3.3 eingeführten UML-Modell lassen sich jedoch auch allgemeinere Regeln anwenden, die auf dieser Ebene definiert worden sind und ermöglichen damit eine Wiederverwendung.

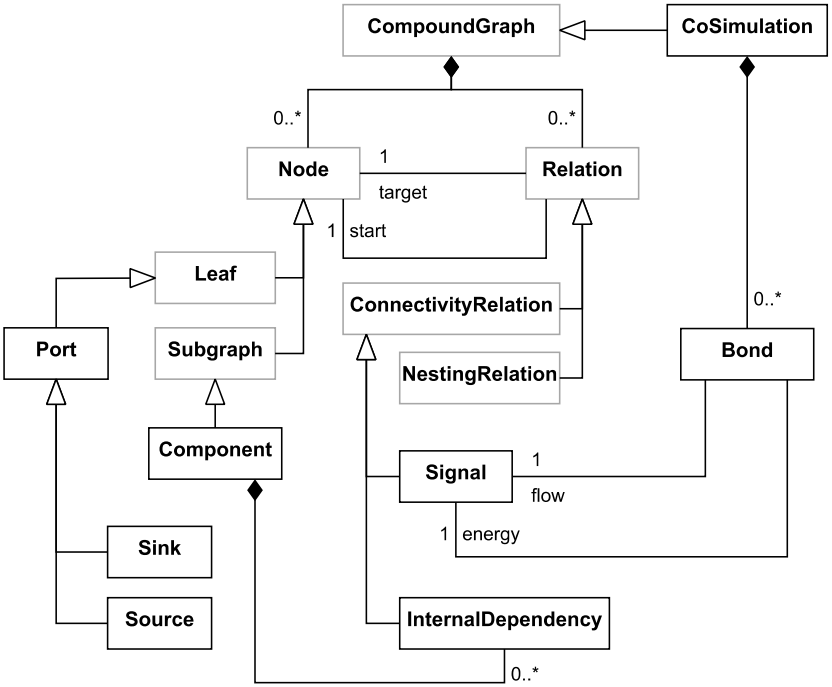


Abbildung 6.9: Domänenspezifische Spezialisierung der Abbildung 4.8 auf die Co-Simulation

6.4 Testfälle innerhalb der Implementierung

Die Implementierung von R43ples erfolgt innerhalb eines Continuous Integration Framework (CIF). Basis bildet hierfür ein GitHub-Repository und Travis CI[1)]. Innerhalb dieser Umgebung sind der Quellcode aber auch Datenbestände wie Ontologien und Testdatensätze vollständig revisioniert. Die umgesetzten Testfälle setzen sich aus Unit Tests, Integration Tests und System Tests zusammen. Integration Tests sind unter anderem für komplexe Anfragen in Bezug auf die Zusammenführung von divergierten Entwicklungszweigen, aber auch für die automatisierte Erzeugung von Beispieldatensätzen umgesetzt.

Zu den Integration Tests gehören des Weiteren auch die Testfälle für die Aggregation von High-Level-Changes und die zugehörige Co-Evolution. Grundlage hierfür bilden zwei Beispieldatensätze (*SampleDataSet.createSampleDataSetHLCAggregation()* und *SampleDataSet.createSampleDataSetCoEvolution()*). Das *SampleDataSetHLCAggregation* stellt einen Revisionsgraphen bereit, der aus einem initialen Commit und einem

[1)] https://travis-ci.com/ (besucht am 29.11.2020)

weiteren Commit besteht. Darin ist exemplarisch die Umbenennung einer Klasse durch die Änderungen von Revision 1 auf Revision 2 beschrieben. Innerhalb vom *SampleDataSetCoEvolution* existiert nur ein initialer Commit. Dieser Commit verwendet jedoch die Klassendefinition aus Revision 1 von *SampleDataSetHLCAggregation* zur Beschreibung der enthaltenen Instanzen. Für die Aggregation beziehungsweise die Co-Evolution wird, wie in den Abschnitten 5.2.3 und 5.3 beschrieben, jeweils ein Regelsatz benötigt. Hierfür wurden die in Listing 16 und Listing 17 dargestellten SPARQL-Anfragen in semantische Beschreibungen der Regelsätze überführt. Der Test *AggregationDraftTest* stellt die korrekte Funktionsweise der durchgeführten Aggregation sicher indem die resultierenden *SemanticChange*s geprüft werden. Im Test *CoEvolutionDraftTest* werden die durchgeführten Co-Evolutionen überprüft. Dabei werden sowohl die semantische Beschreibung der Co-Evolution als auch die Anpassungen am abhängigen Revisionsgraphen getestet.

Bei den Tests des Gesamtsystems kommt als Triple Store Jena TDB[2)] zum Einsatz. Anfragen an R43ples werden in diesen Testfällen nicht über interne Schnittstellen gestellt, sondern über die gleiche Schnittstelle, die später durch Nutzer verwendet werden.

[2)] https://jena.apache.org/documentation/tdb/ (besucht am 29.11.2020)

7 Diskussion

Im Rahmen dieser Arbeit wurde ein RMS zur Unterstützung der Evolution von Informations- und Datenmodellen entwickelt, das Revisionsverwaltungs- und Evolutionsmechanismen integriert. Besonderheit ist hierbei die technologieunabhängige mathematische und semantische Beschreibung, die eine Überführung des Konzepts in unterschiedliche Technologien ermöglicht. Im Folgenden wird die Methodik der Arbeit bewertet und anschließend werden die Ergebnisse den aufgestellten Thesen gegenübergestellt, um diese zu verifizieren.

7.1 Methodikbewertung

Ausgangspunkt dieser Arbeit bildete eine detaillierte Anforderungsanalyse, die sich auf einer Literaturrecherche stützt. Da mit diesem Vorgehen die Gefahr einhergeht, dass wichtige Quellen nicht betrachtet werden oder Anforderungen in der Realität nicht zum Tragen kommen, wurde ein strukturiertes Vorgehen angewendet, um dem entgegenzuwirken. Gleichwohl erhebt diese Arbeit keinen Anspruch auf Vollständigkeit der aufgenommenen Anforderungen. Ein wesentliches Element war von Beginn an die gemeinsame Betrachtung von Evolution und Revisionsverwaltung. Ausgehend von allgemeinen Prinzipien mit Einfluss auf Evolvability wurde festgestellt, dass etablierte Systeme in diesem Bereich an ihre Grenzen stoßen und neue integrierte Mechanismen geschaffen werden müssen. Die allgemeinen Prinzipien wurden dann im Anschluss auf eine technologische Sicht gehoben und durch weitere Literatur untermauert. Die sich daraus ergebenden Kriterien wurden wiederum den Anwendungsfällen der Arbeit gegenübergestellt, wodurch gezeigt werden konnte, dass diese Anforderungen praktische Relevanz besitzen. Im Folgenden wurden sie bestehenden Ansätzen gegenübergestellt, um bereits vorhandene Aspekte zu identifizieren, die in einem integrierten Konzept wiederverwendet werden können. Durch die anschließende Priorisierung wurde der Fokus der Arbeit auf die Kernanforderungen gelegt, wodurch diese im Detail im Rahmen dieser Arbeit ausgearbeitet werden konnten. Darauf aufbauende weiterführende Kriterien müssen in folgenden Arbeiten weiter detailliert und in Konzepte überführt werden. Das in dieser Arbeit beschriebene Konzept sieht demnach eine Struktur des RMS vor, die es ermöglicht, auch die niedriger priorisierten Anforderungen in Folgearbeiten zu integrieren. Die im Entwurf dargestellten technologieunabhängigen mathematischen und semantischen Beschreibungen wurden innerhalb einer prototypischen Implementierung für das Semantic Web angewendet. Durch die Weiterentwicklung von dem bestehenden semantischen Revisionskontrollsystem R43ples zu einem RMS konnte die Funktionsweise des Konzeptes nachgewiesen werden. Dies wurde in einer anschließenden Verifikation auch mittels theoretischer und praktischer Tests nachgewiesen. Eine Übertragung auf andere Technologien, wie zum Beispiel OPC UA, wurde im Rahmen dieser Arbeit nicht untersucht. Erste Arbeiten in

diesem Bereich zeigen jedoch, dass eine bidirektionale Synchronisierung zwischen Semantic Web und OPC UA möglich ist [Ahr18]. Hieraus kann abgeleitet werden, dass auch die Mechanismen dieser Arbeit innerhalb von OPC UA anwendbar sind. Die Bestätigung oder die Widerlegung dieser weiterführenden These muss jedoch in folgenden Arbeiten detailliert untersucht werden. Bei der im Rahmen dieser Arbeit zur Verfügung gestellten Implementierung handelt es sich um einen Prototypen eines RMS für das Semantic Web. Trotz der Verwendung eines CIF und der Erstellung von Testfällen muss das System eine Weiterentwicklung erfahren, bevor es in einen Produktiveinsatz überführt werden kann. Auf der einen Seite muss hierfür die Testabdeckung weiter erhöht werden, aber auf der anderen Seite müssen auch die visuellen Interaktionsmöglichkeiten mit dem System ausgebaut werden, um die Nutzerakzeptanz zu erhöhen.

7.2 Ergebnisdiskussion und Verifikation der Thesen

Zielstellung dieser Arbeit war die durchgängige Unterstützung der Evolution von Informations- und Datenmodellen über deren Lebenszyklus hinweg. Dieses Ziel konnte durch Lebenszyklusbetrachtungen und die Konzeption eines RMS erreicht werden. Das RMS bildet die Grundlage, Evolutionen über den gesamten Lebenszyklus hinweg zu unterstützen. Kernergebnis dieser Arbeit ist dabei die technologieunabhängige integrierte mathematische und semantische Beschreibung von Revisionsverwaltung und Evolution für Informations- und Datenmodelle sowie das Konzept eines RMS. Die bereitgestellten Funktionen für die Basisrevisionsverwaltung ermöglichen hierbei die Interaktion mit dem Repository. Im Rahmen dieser Arbeit wurde insbesondere auf die Funktionen zur Erstellung von Commits, Branches und Tags eingegangen. Mittels der Revert-Funktion können Commits rückgängig gemacht werden, bleiben aber in der Revisionshistorie weiterhin erhalten. Für Branches und Tags stehen Funktionen zur Löschung nicht bereit, können aber als Umkehroperation aus den Funktionen für die Erstellung heraus abgeleitet werden. Aufgrund der technologieunabhängigen Beschreibung mussten innerhalb von den mathematischen Beschreibungen Schnittstellen geschaffen werden, die eine technologieabhängige Umsetzung ermöglichen. Dies tritt unter anderem bei der Aggregation von atomaren Änderungen hin zu High-Level-Changes auf. Regelsätze und Mechanismen der Aggregation sind dabei weitestgehend technologieabhängig und dementsprechend ausgegliedert worden. An diesen Stellen existieren jedoch zumeist bereits Lösungen, wie unter anderem [Keh15] oder [Pap+13], auf die bei einer Umsetzung zurückgegriffen werden kann. Für die Zusammenführung von divergierten Entwicklungszweigen stehen im Rahmen dieser Arbeit drei unterschiedliche Möglichkeiten zur Verfügung. Da bei der Zusammenführung Konflikte auftreten können, wurden hierfür Mechanismen entwickelt, um diese Konflikte auf einer strukturellen Ebene zu lösen beziehungsweise transiente Effekte auf der High-Level-Ebene zu detektieren. Im Rahmen der Implementierung wurden jedoch nur die Mechanismen der strukturellen Analyse umgesetzt. Die Durchführung von Co-Evolutionen basiert auf den aggregierten High-Level-Changes sowie den Basisrevisionskontrollfunktionalitäten. Durch eine Analyse der Abhängigkeiten innerhalb des RMS können dann entsprechende Co-Evolutionen vorgenommen werden. Diese

basieren innerhalb der Implementierung jedoch nur auf Typ-Instanz-Ebene und nicht auf der Beschreibung von Verbindungsgraphen, die Abhängigkeiten zwischen Datenmodellen abbilden. Die Modellierung der Compound Graphs wurde jedoch anhand des Anwendungsfalls der Co-Simulation theoretisch verifiziert.

Rückblickend auf die der Arbeit zugrunde gelegten Thesen kann an dieser Stelle festgestellt werden, dass diese im Rahmen der Arbeit positiv beantwortet werden konnten. Nachfolgend werden die einzelnen wissenschaftlichen Forschungsthesen aus der Arbeit heraus begründet:

These 1: *Neue Anforderungen an die Agilität von Produktlebenszyklen erfordern Veränderungen im Lebenszyklus der zugrundeliegenden Informationsräume, vor allem im Bereich der Revisionierung und Evolution der Informations- und Datenmodelle.*

Diese These konnte durch die durchgeführte Literaturrecherche und die Betrachtung der Anwendungsfälle bestätigt werden. Individualisierung sowie kürzere Produktlebenszyklen führen zu Veränderungen in der Produktion, aber auch in der Standardisierung von Schnittstellen. In diesen Bereichen stellen sich neue Herausforderungen an die Agilität, mit der auf Anforderungsänderungen reagiert werden muss, um weiterhin wettbewerbsfähig zu bleiben. Neue Ansätze, wie zum Beispiel die Modularisierung und die damit verbundenen Standardisierungsarbeiten, bieten hierfür Lösungen an. Sie stellen dabei aber wiederum neue Anforderungen an die Revisionierung und Evolution von Informationsräumen, da sich aufgrund der Agilität der Standardisierung auch die zugehörigen Schnittstellen weiterentwickeln und mit diesen Änderungen umgegangen werden muss. Durch Ansätze wie [Gra16] wird des Weiteren eine semantische Integration von Informationsräumen ermöglicht, durch die unterschiedliche Modelle miteinander gekoppelt werden können. Auch diese neu entstehenden Abhängigkeiten stellen neue Anforderungen an die Revisionierung und die Evolution der Modelle.

These 2: *Anforderungen können durch etablierte Werkzeuge aus der Softwareentwicklung nicht vollständig erfüllt werden.*

Zur Bestätigung dieser These wurden die aufgenommenen Anforderungen gegen bestehende Ansätze verglichen, woraus sich ergeben hat, dass diese die Anforderungen nicht vollständig erfüllen können und auch diese These als bestätigt angesehen werden kann. Für die Revisionsverwaltung gibt es zwar etablierte Ansätze, die aber aufgrund der Struktur und der Natur von Informations- und Datenmodellen nicht genutzt werden können. Diese Systeme agieren nicht auf einer Inhaltsebene, sondern auf einer zeilenbasierten Strukturebene, die für die Revisionierung von Modellen nicht zielführend ist. Ebenso existieren Ansätze für die Evolution von Modellen, wobei diese jedoch überwiegend den Aspekt der Revisionierung außen vorlassen, wodurch die Nachvollziehbarkeit der durchgeführten Änderungen nicht mehr gewährleistet ist.

These 3: *Die Integration von Revisionskontrollfunktionalitäten und Evolutionsmechanismen in ein übergeordnetes Revision Management System bietet die Grundlage für die Umsetzung der Anforderungen.*

Aus der Anforderungsanalyse heraus hat sich ergeben, dass die Integration von Revisionskontrollfunktionalitäten und Evolutionsmechanismen notwendig ist, um die Anforderungen umzusetzen. Hierfür wurde ein RMS konzeptioniert, das aus unterschiedlichen Komponenten aufgebaut ist, die jeweils die Umsetzung von spezifischen aufgenommenen Anforderungen erlauben. Dadurch kann diese These bestätigt werden. In dieser Arbeit wurden die grundlegenden Komponenten im Detail ausspezifiziert, wobei die mathematischen und semantischen Beschreibungen der jeweiligen Komponenten aufeinander aufbauen. Ein wesentliches Merkmal ist dabei die semantische Beschreibung über alle Komponenten hinweg. Dies ermöglicht es einerseits, auf Ergebnisse von anderen Komponenten zuzugreifen, sowie andererseits die Möglichkeit der Umsetzung von weiterführenden Komponenten und zukünftigen Erweiterungen.

These 4: *Die technologieunabhängige Beschreibung des Revision Management Systems erlaubt eine Umsetzung in unterschiedlichen Anwendungsdomänen.*

Diese These wurde durch die Umsetzung der technologieunabhängigen Beschreibung für das Semantic Web und die anschließende Verifikation bestätigt. Hierbei konnte gezeigt werden, dass die technologieunabhängigen UML-Modelle in Ontologien für das Semantic Web überführt werden konnten. Des Weiteren zeigen erste Arbeiten, dass eine bidirektionale Synchronisation zwischen Semantic Web und OPC UA möglich ist, woraus ableitbar ist, dass ein RMS ebenso für OPC UA umsetzbar ist. Hierbei müssen die in UML beschriebenen Informationsmodelle in Typenmodelle von OPC UA überführt werden.

Kernthese *Ein Revision Management System unterstützt die Evolution von Informations- und Datenmodellen über deren gesamten Lebenszyklus durch die Integration von Revisionskontroll- und Evolutionsmechanismen.*

Durch die vorangegangene Bestätigung der Einzelthesen kann ebenso die Kernthese dieser Arbeit als bestätigt angesehen werden. Innerhalb der Arbeit wurde ausgehend von einem Lebenszyklusmodell für Informationsmodelle ein RMS entwickelt, das Revisionskontroll- und Evolutionsmechanismen integriert und damit die Grundlage für eine durchgängige Lebenszyklusunterstützung von Informations- und Datenmodellen schafft.

8 Zusammenfassung

Neben einer kurzen Ergebniszusammenfassung der Arbeit werden im Folgenden Anknüpfungspunkte aufgezeigt, an denen folgende Arbeiten ansetzen können.

8.1 Ergebniszusammenfassung

Ausgehend von einer Literaturrecherche und der daraus abgeleiteten Anforderungsanalyse wurde innerhalb dieser Arbeit ein Lebenszyklusmodell für Informationsmodelle sowie ein RMS entwickelt, das Revisionskontroll- und Evolutionsmechanismen technologieunabhängig integriert. Dieses RMS bietet damit die Grundlage für eine durchgängige Lebenszyklusunterstützung von Informations- und Datenmodellen in unterschiedlichen Anwendungsdomänen. Durch eine Umsetzung des Konzeptes im Semantic Web als eine Weiterentwicklung des Open-Source-Projektes R43ples und die anschließende Verifikation konnte nachgewiesen werden, dass das Konzept funktionsfähig ist und die notwendigen Funktionen für die Lebenszyklusunterstützung bereitstellt.

8.2 Ausblick und Grenzen

Die sich aus dieser Arbeit ergebenden Anknüpfungspunkte für folgende Arbeiten beziehungsweise Fragestellungen, die in weiteren Arbeiten untersucht werden müssen, sind nachfolgend in drei Kategorien unterteilt. Im ersten Schritt werden offene Punkte mit Bezug zum vorgestellten Konzept aufgezeigt. Anschließend werden Weiterentwicklungsmöglichkeiten skizziert, die sich aus der Implementierung im Semantic Web schlussfolgern lassen. Schließlich werden offene Fragestellungen aufgeschlüsselt, die sich aus der gesamten Arbeit heraus ergeben beziehungsweise innerhalb dieser Arbeit nicht mehr betrachtet werden konnten.

Konzept In Bezug auf das Konzept bestehen offene Punkte vor allem in Hinblick auf die vorgenommene Priorisierung und die damit verbundenen nicht im Detail beschriebenen Komponenten eines RMS. Hierzu gehört zum einen die Bereitstellung eines User Interfaces für die nutzerfreundliche Interaktion mit dem System und zum anderen das Zugriffsmanagement. Innerhalb dessen müssen unter anderem Freigabeprozesse für Informationsmodelle im Idealfall anhand industrieller Anwendungsszenarien näher untersucht werden. Hierbei müssen weiterhin die Rollen innerhalb einer solchen Umgebung im Detail erfasst werden und sichergestellt werden, welche Änderungen von wem nachverfolgt werden müssen. Dabei wird außerdem Wissen benötigt, wo und von wem Informationsmodelle für die Beschreibung von Datenmodellen eingesetzt werden. Dies ist notwendig,

da diese Personengruppen bei einer möglichen Außerkraftsetzung entsprechend informiert und daraus weiterführende Maßnahmen abgeleitet werden müssen.

Wie bereits in der Diskussion aufgeführt, werden weitere Funktionen als Umkehroperationen der Erstellungsfunktionen benötigt, um zum Beispiel einen Branch oder einen Tag rückgängig zu machen. Ebenso können in zukünftigen Arbeiten die Mechanismen der High-Level-Change-Aggregation auf einen ganzen Revisionspfad ausgedehnt werden, um entlang von diesem Pfad alle High-Level-Changes zu detektieren und auch Abhängigkeiten zwischen diesen aufzulösen. Da in dieser Arbeit Schnittstellen zu technologiespezifischen Lösungen geschaffen werden mussten, müssen diese Schnittstellen für die jeweiligen Technologien spezifisch umgesetzt werden. Hier könnte in Folgearbeiten ebenso untersucht werden, ob es Überschneidungen zwischen unterschiedlichen Technologien gibt und Teile der Realisierung auf eine unabhängige Ebene gehoben werden können.

Ein weiterer konzeptionell zu betrachtender Anwendungsfall besteht in der Wiederverwendbarkeit von Teilen von abgelösten Informationsmodellen in neu erstellten Informationsmodellen und wie in diesem Szenario Migrationen durchgeführt werden können, um Konsistenz sicherzustellen. Dabei tritt des Weiteren die Notwendigkeit der Referenzierung zwischen unterschiedlichen Revisionen innerhalb der Informations- und Datenmodelle auf. Diese wird durch die unterschiedlichen Ausprägungen der Beschreibung der Modelle jedoch mit hoher Wahrscheinlichkeit technologiespezifisch ausfallen. Hierfür müssen die notwendigen mathematischen und semantischen Grundlagen geschaffen werden.

Implementierung Die Implementierung ist eine prototypische Umsetzung des vorgestellten Konzeptes, woraus sich ebenso offene Punkte ergeben, die zukünftig weiterentwickelt beziehungsweise untersucht werden müssen. Innerhalb von R43ples wurden beispielsweise die Aggregations- und Co-Evolutionsfunktionen nur für zwei direkt aufeinanderfolgende Revisionen umgesetzt. An dieser Stelle muss eine Erweiterung hin zu der Aggregation und Co-Evolution entlang eines Pfades von Revisionen stattfinden, um auch mehrere durchgeführte Änderungen auf einmal co-evolvieren zu können. Im Fall, dass entlang eines Pfades co-evolviert werden soll, der nur durch eine Start- und eine Zielrevision gekennzeichnet ist, können potenziell unterschiedliche Pfade im Revisionsgraphen genutzt werden. Hier muss untersucht werden, inwieweit bereitgestellte Regelsätze unabhängig vom Pfad sind, oder ob der gewählte Pfad Einfluss auf die Co-Evolutionen hat. Ebenso ist der bereitgestellte Regelsatz für Aggregation und Co-Evolution nur für eine Beispielregel umgesetzt. Die Integration von weiteren Regeln ist essenziell für eine breite Anwendung der Implementierung. Die Regeln beziehen sich außerdem auf Typ-Instanz-Beziehungen und müssen mit weiteren Regeln in Bezug auf die noch umzusetzende Komponente eines *ConnectionManager*s erweitert werden, wobei diese Regeln einerseits auf den allgemeinen Definitionen von Compound Graphs aber auch auf anwendungsspezifischen Erweiterungen aufbauen sollten. Des Weiteren muss die Zusammenführung von divergierten Entwicklungszweigen um die vorgestellten Möglichkeiten der semantischen Konfliktdetektion erweitert werden. Verteilte Anfragen, die mehrere revisionierte Modelle auf einmal ändern, werden von der aktuellen Implementierung nicht unterstützt. An dieser Stelle muss in weiteren Arbeiten untersucht werden, wie sich gleichzeitige Änderungen auf die

durchzuführenden Co-Evolutionen auswirken. Beispielsweise kann daraus die Notwendigkeit entstehen, dass eine semantische Verknüpfung zwischen den einzelnen erstellten Commits beziehungsweise den erstellten Revisionen herzustellen ist. Schließlich sollte die gesamte Performance der Implementierung gesteigert werden. Hierfür können beispielsweise Mechanismen untersucht werden, die eine Selbstoptimierung des Repositories in Bezug auf kürzeste Wege und vollständig vorgehaltenen Revisionen ermöglichen.

R43ples verfügt aktuell über ein Webinterface, über das Anfragen an das System gestellt werden können und das auch rudimentäre Information zu den revisionierten Modellen bereitstellt. Jedoch fehlen Visualisierungsmöglichkeiten von zum Beispiel Differenzen zwischen Revisionen und eine Aufbereitung der erkannten High-Level-Changes. Ebenso sollte zukünftig die Durchführung von Co-Evolutionen visuell unterstützt werden. Erste Ansätze wurden im Rahmen einer studentischen Arbeit [Fun17] entwickelt, die in die prototypische Implementierung überführt werden könnten. Im Zusammenhang mit der Co-Evolution müssen weiterhin die Verbindungen zwischen Modellen gepflegt werden. Hierfür werden ebenfalls Benutzerschnittstellen benötigt, die bestehende Verbindungen darstellen und eine nachträgliche Bearbeitung erlauben. Die Erstellung der Regelsätze erfolgt im aktuellen Ansatz manuell, was mit einer Fehleranfälligkeit und hohen zeitlichen Aufwänden verbunden ist. Diesem Problem könnte auch durch eine geeignete visuelle Konfigurationsschnittstelle entgegengewirkt werden.

Offene Fragestellungen für weiterführende Arbeiten Die vorliegende Arbeit nutzt zwei Anwendungsfälle mit praktischer Relevanz als Grundlage. Gleichwohl sollten sowohl die Implementierung als auch das Konzept im Allgemeinen im industriellen Umfeld anhand konkreter industrieller Anwendungsfälle weiter evaluiert werden. An dieser Stelle könnte ebenso eine Übertragung des Konzeptes in eine andere Technologie wie zum Beispiel OPC UA stattfinden. Darüber hinaus könnte das vorgestellte RMS auch die Grundlage für ein CIF für Informations- und Datenmodelle bereitstellen. Innerhalb von diesem wäre es dann zum Beispiel auch möglich, Analysen über Wiederverwendung oder Qualität von Informationsmodellen durchzuführen, wodurch die semantische Beschreibung innerhalb von Informationsräumen verbessert werden könnte. In diesem Zusammenhang ist auch die Entwicklung eines semantischen Ticketsystems vorstellbar, das sich auf das RMS bezieht.

Wie bereits angesprochen, müssen die Regelsätze derzeit mit hohem manuellem Aufwand erstellt werden. In weiterführenden Arbeiten könnte untersucht werden, inwieweit wiederkehrende Änderungsmuster automatisiert erkannt werden können, um dann beispielsweise nur noch durch einen Nutzer mit einer entsprechenden Semantik versehen werden zu müssen. Darüber hinaus kann das vorgestellte RMS auch innerhalb von wissensbasierten Systemen für das Round-Trip-Engineering als Grundlage für die Revisionierung und Evolution genutzt werden. Hierbei müssen auch Methodiken entwickelt werden, um Anfragen auf Modelle beziehungsweise Abbildungsvorschriften zwischen Modellen mit den Modellen zu evolvieren.

Mit der Umsetzung im Semantic Web erfolgte ein erster Schritt für die durchgängige Unterstützung der Evolution von vernetzten Modellen. Jedoch muss in folgenden Ar-

beiten auch die Evolution über Repository-Grenzen hinweg untersucht werden, da die entsprechenden Modelle zwar Abhängigkeiten besitzen, aber nicht zwangsläufig innerhalb einer Umgebung revisioniert werden, sondern potenziell global verteilt sein können. Darüber hinaus sollten Bestrebungen aufgenommen werden, SPARQL-Erweiterungen für die Revisionsverwaltung und die Evolution in eine Standardisierung zu überführen, um einen einheitlichen Zugriff über Applikationen hinweg zu ermöglichen.

Anhang

A Entwurf

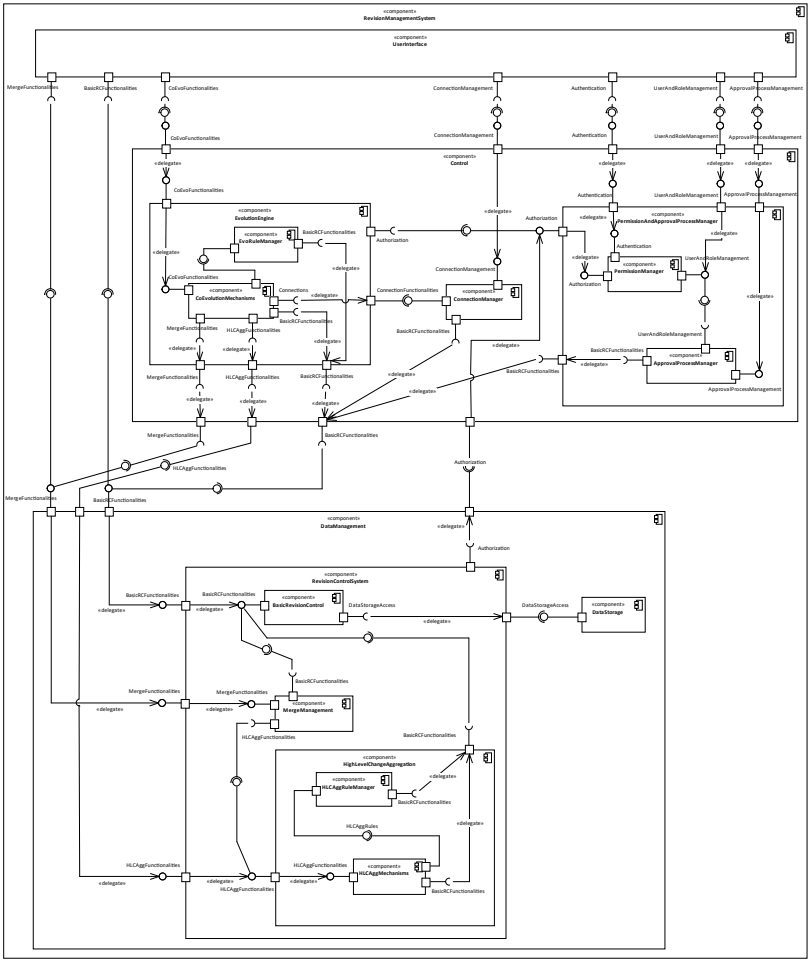


Abbildung A.1: Vollständiges Komponentendiagramm des RMS

B Implementierung

Im Nachfolgenden wird anhand von konkreten Beispielen dargestellt, wie sich der Revisionsraum verändert, wenn bereitgestellte Funktionalitäten mittels der erweiterten SPARQL-Syntax ausgeführt werden. In den Bildern stehen dünne Linien für bestehende Information, dicke Linien für neu hinzugekommene Information und gepunktete Linien kennzeichnen entfernte Information. Revisionen werden aus Gründen der Übersichtlichkeit grau hervorgehoben.

B.1 Basisrevisionskontrollfunktionalitäten

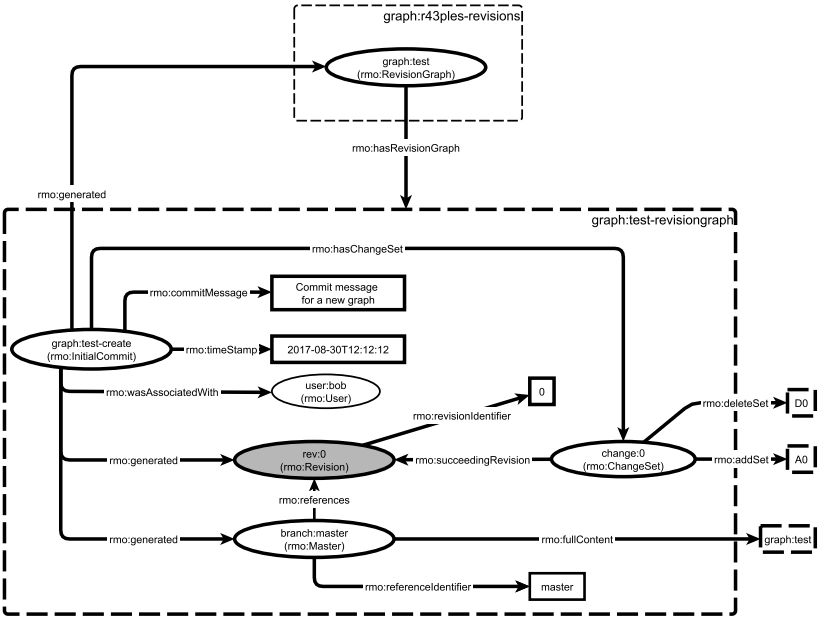


Abbildung B.1: Beispiel für die Erstellung eines neuen Graphen

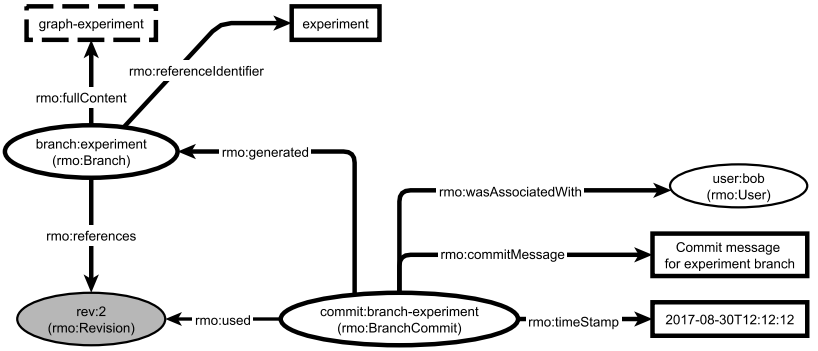


Abbildung B.2: Beispiel für die Erstellung eines neuen Entwicklungszweiges

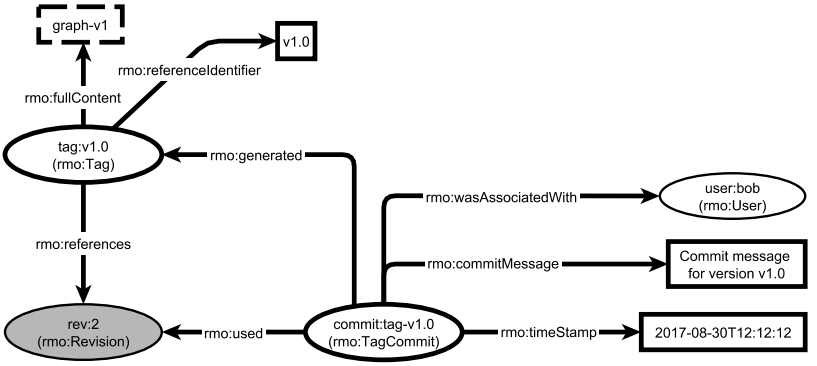


Abbildung B.3: Beispiel für die Erstellung eines neuen Tags

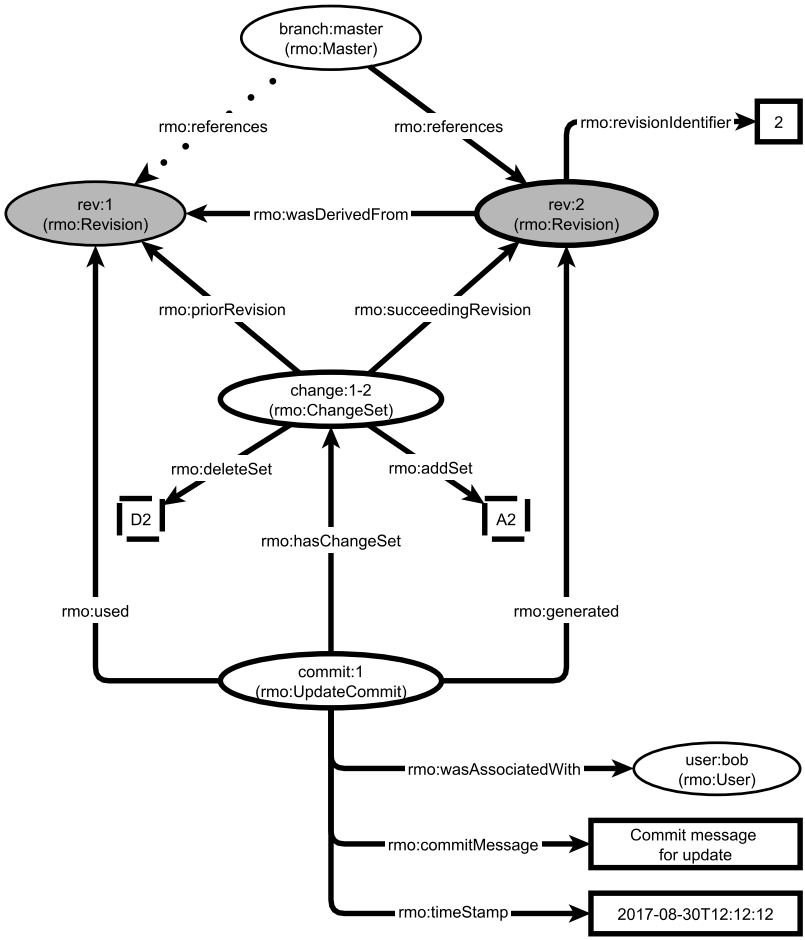


Abbildung B.4: Beispiel für die Erstellung eines neuen Commits

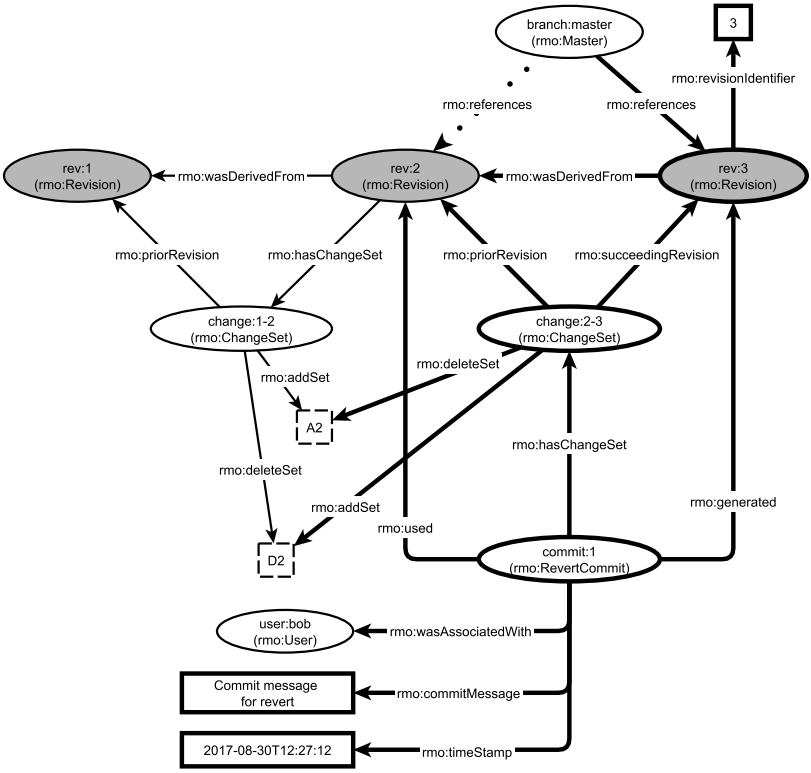


Abbildung B.5: Beispiel für die Revidierung eines Commits

B.2 Aggregation von High-Level-Changes

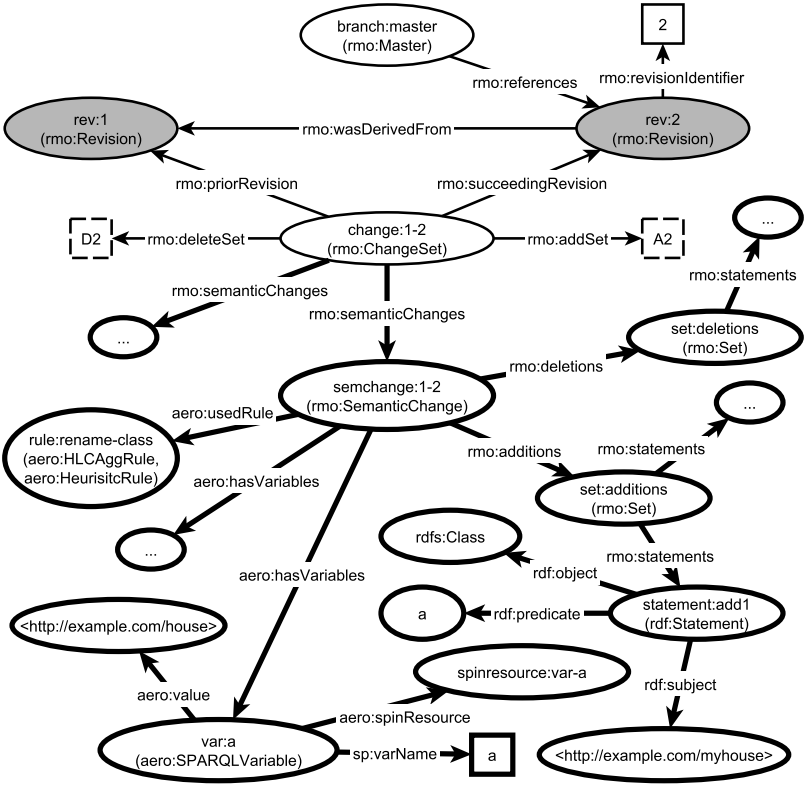


Abbildung B.6: Beispiel für die Aggregation (Ausschnitt)

```
1  PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
2  PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3
4  SELECT ?b ?c ?resource
5  WHERE {
6    {
7      SELECT ?b ?c ?resource
8      WHERE {
9        GRAPH <http://NAMEDGRAPH#ADDSET-1-2> {   # Example: <http://
             NAMEDGRAPH#ADDSET-1-2> will be replaced with <http://test.com/
             r43ples-dataset-hlc-aggregation-addSet-1-2>
10         ?c a rdfs:Class.
11         ?c rdfs:subClassOf ?resource.
12       }
13     }
14   }
15   {
16     SELECT ?b ?c ?resource
17     WHERE {
18       GRAPH <http://NAMEDGRAPH#DELETESET-1-2> {   # Example: <http://
             NAMEDGRAPH#DELETESET-1-2> will be replaced with <http://test.
             com/r43ples-dataset-hlc-aggregation-deleteSet-1-2>
19         ?b a rdfs:Class.
20         ?b rdfs:subClassOf ?resource.
21       }
22     }
23   }
24   MINUS
25   {
26     SELECT ?b ?c
27     WHERE {
28       GRAPH <http://NAMEDGRAPH#rev1> {  # Example: <http://NAMEDGRAPH#
             rev1> will replaced with <http://test.com/r43ples-dataset-hlc-
             aggregation> REVISION "1"
29         ?c ?s1 ?o1.
30       }
31       GRAPH <http://NAMEDGRAPH#rev2> {  # Example: <http://NAMEDGRAPH#
             rev2> will replaced with <http://test.com/r43ples-dataset-hlc-
             aggregation> REVISION "2"
32         ?b ?s2 ?o2.
33       }
34     }
35   }
36 }
```

Listing 16: Beispiel für die in SPIN beschriebene Anfrage

B.3 Zusammenführung von divergierten Entwicklungszweigen

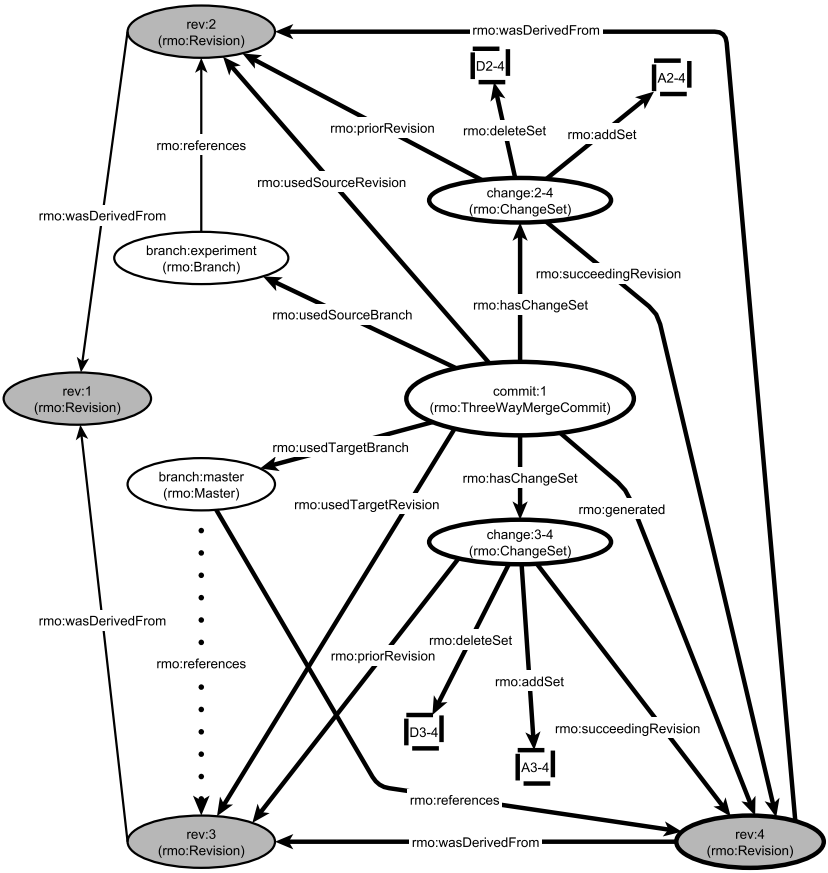


Abbildung B.7: Beispiel für einen 3-Wege-Merge Commit

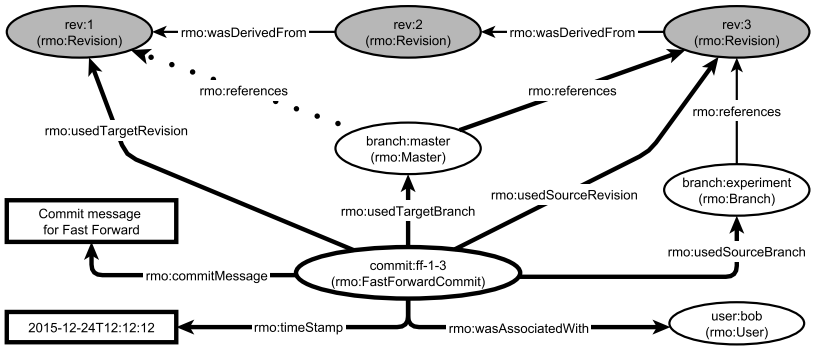


Abbildung B.8: Beispiel für einen Fast Forward Commit

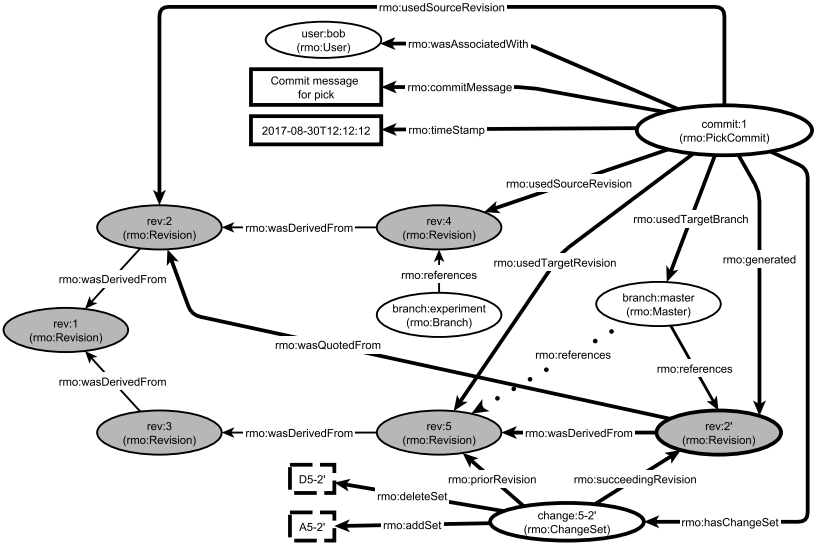


Abbildung B.9: Beispiel für einen Pick Commit

B.4 Co-Evolution

```
1  # Matching query:
2  SELECT ?subject
3  WHERE {
4    GRAPH <http://NAMEDGRAPH#master> { # Example: <http://NAMEDGRAPH#
         master> will replaced with <http://test.com/r43ples-dataset-1>
5      ?subject a ?b .
6    }
7  }
8
9  # ADD set update query:
10 INSERT
11   { GRAPH <http://NAMEDGRAPH#ADDSET-NEW> { ?subject a ?c } } # Example:
       <http://NAMEDGRAPH#master> will replaced with <http://test.com/
       r43ples-dataset-1-addSet-4-5>
12 WHERE
13   { GRAPH <http://NAMEDGRAPH#master> # Example: <http://NAMEDGRAPH#
       master> will replaced with <http://test.com/r43ples-dataset-1>
14     { ?subject a ?b .
15     }
16   }
17
18 # DELETE set update query
19 INSERT
20   { GRAPH <http://NAMEDGRAPH#DELETESET-NEW> { ?subject a ?b } } #
       Example: <http://NAMEDGRAPH#master> will replaced with <http://
       test.com/r43ples-dataset-1-deleteSet-4-5>
21 WHERE
22   { GRAPH <http://NAMEDGRAPH#master> # Example: <http://NAMEDGRAPH#
       master> will replaced with <http://test.com/r43ples-dataset-1>
23     { ?subject a ?b .
24     }
25   }
```

Listing 17: Beispiel für die in SPIN beschriebenen Anfragen zur Detektion von Matchings und zur Spezifikation der Co-Evolution

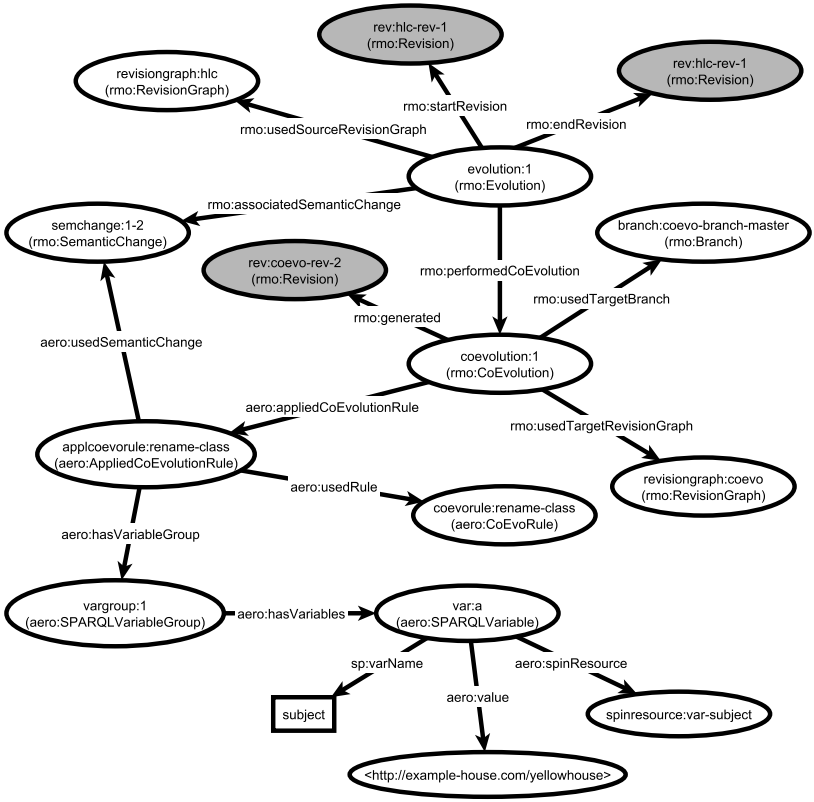


Abbildung B.10: Beispiel für die Co-Evolution (Ausschnitt)

Literaturverzeichnis

[AH06] Sören Auer und Heinrich Herre. „A Versioning and Evolution Framework for RDF Knowledge Bases“. en. In: *Perspectives of Systems Informatics*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Juni 2006, S. 55–69. ISBN: 978-3-540-70880-3 978-3-540-70881-0. DOI: 10.1007/978-3-540-70881-0_8. URL: https://link.springer.com/chapter/10.1007/978-3-540-70881-0_8 (besucht am 29. 11. 2020) (siehe S. 96).

[Ahr18] Michael Ahrens. „Entwicklung einer Middleware zur Integration und Synchronisation von OPC UA und Semantic Web Informationsmodellen“. Diplomarbeit. Dresden: TU Dresden, Nov. 2018 (siehe S. 118, 133).

[AM17] Natanael Arndt und Michael Martin. „Decentralized Evolution and Consolidation of RDF Graphs“. In: *17th International Conference on Web Engineering (ICWE 2017)*. ICWE 2017. Rome, Italy, Juni 2017. DOI: 10.1007/978-3-319-60131-1_2. URL: https://svn.aksw.org/papers/2017/ICWE_DecentralizedEvolution/public.pdf (besucht am 29. 11. 2020) (siehe S. 3, 27, 72, 73, 83).

[ARM16] Natanael Arndt, Norman Radtke und Michael Martin. „Distributed Collaboration on RDF Datasets Using Git: Towards the Quit Store“. In: *Proceedings of the 12th International Conference on Semantic Systems*. SEMANTiCS 2016. New York, NY, USA: ACM, 2016, S. 25–32. ISBN: 978-1-4503-4752-5. DOI: 10.1145/2993318.2993328. URL: http://doi.acm.org/10.1145/2993318.2993328 (besucht am 29. 11. 2020) (siehe S. 27).

[Aue+12] Sören Auer, Lorenz Bühmann, Christian Dirschl, Orri Erling, Michael Hausenblas, Robert Isele, Jens Lehmann, Michael Martin, Pablo N. Mendes, Bert van Nuffelen, Claus Stadler, Sebastian Tramp und Hugh Williams. „Managing the Life-Cycle of Linked Data with the LOD2 Stack“. en. In: *The Semantic Web – ISWC 2012*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Nov. 2012, S. 1–16. ISBN: 978-3-642-35172-3 978-3-642-35173-0. DOI: 10.1007/978-3-642-35173-0_1. URL: https://link.springer.com/chapter/10.1007/978-3-642-35173-0_1 (besucht am 29. 11. 2020) (siehe S. 59).

[Bae05] Stefan Baerisch. *Versionskontrollsysteme in der Softwareentwicklung*. Arbeitsbericht / Informationszentrum Sozialwissenschaften Nr. 36. Bonn: IZ Sozialwissenschaften, 2005 (siehe S. 23).

[Ban+87] Jay Banerjee, Won Kim, Hyoung-Joo Kim und Henry F. Korth. „Semantics and Implementation of Schema Evolution in Object-oriented Databases“. In: *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*. SIGMOD '87. New York, NY, USA: ACM, 1987, S. 311–322. ISBN: 978-0-89791-236-5. DOI: 10.1145/38713.38748. URL: http://doi.acm.org/10.1145/38713.38748 (besucht am 29. 11. 2020) (siehe S. 20).

[Bas+11] Jens Bastian, Christoph Clauß, Susann Wolf und Peter Schneider. „Master for Co-Simulation Using FMI“. In: Juni 2011, S. 115–120. DOI: 10.3384/ecp11063115. URL: https://ep.liu.se/ecp/063/014/ecp11063014.pdf (besucht am 29. 11. 2020) (siehe S. 41).

[BB08] A. Terry Bahill und Rick Botta. „Fundamental Principles of Good System Design“. In: *Engineering Management Journal* 20.4 (Dez. 2008). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.8459&rep=rep1&type=pdf (besucht am 29. 11. 2020) (siehe S. 18, 32–35).

[BBR09] Robert Brcina, Stephan Bode und Matthias Riebisch. „Optimisation Process for Maintaining Evolvability during Software Evolution“. In: *2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. Apr. 2009, S. 196–205. DOI: 10.1109/ECBS.2009.20 (siehe S. 18, 19).

[BCE07] Hongyu Pei Breivold, Ivica Crnkovic und Peter Eriksson. „Evaluating Software Evolvability“. In: *Software Engineering Research and Practice in Sweden* (2007), S. 96 (siehe S. 18).

[BCE08] Hongyu Pei Breivold, Ivica Crnkovic und Peter J. Eriksson. „Analyzing Software Evolvability“. In: *2008 32nd Annual IEEE International Computer Software and Applications Conference*. Juli 2008, S. 327–330. DOI: 10.1109/COMPSAC.2008.50. URL: https://ieeexplore.ieee.org/document/4591576 (besucht am 29. 11. 2020) (siehe S. 20).

[Bee12] Jay Clark Beesemyer. „Empirically characterizing evolvability and changeability in engineering systems“. en. Masterarbeit. Massachusetts Institute of Technology, 2012. URL: http://dspace.mit.edu/handle/1721.1/76092 (besucht am 29. 11. 2020) (siehe S. 18).

[BH17] Jens Bernshausen und Axel Haller. *NAMUR MTP - Visualization and Control of Modular Plants*. Workshop. Bad Neuenahr, Nov. 2017 (siehe S. 6).

[Bie+16] Thomas Bieringer, Christian Bramsiepe, Stefan Brand, Andreas Brodhagen, Christian Dreiser, Christoph Fleischer-Trebes, Norbert Kockmann, Stefan Lier, Dirk Schmalz, Christian Schwede, Armin Schweiger und Frank Stenger.

Modular Plants: Flexible chemical production by modularization and standardization – status quo and future trends (ProcessNet White Paper). Frankfurt am Main: Dechema e.V., Dez. 2016. ISBN: 978-3-89746-191-2. URL: http://dechema.de/dechema_media/ModularPlants_2016-p-20002425.pdf (besucht am 29. 11. 2020) (siehe S. 1, 5).

[Blo+17] Henry Bloch, Stephan Hensel, Mario Hoernicke, Katharina Stark, Anna Menschner, Leon Urbas, Alexander Fay, Torsten Knohl, Jens Bernshausen und Axel Haller. „Zustandsbasierte Führung modularer Prozessanlagen". In: *atp edition* 59.10 (Okt. 2017), S. 34–45. ISSN: 2190-4111. URL: http://ojs.di-verlag.de/index.php/atp_edition/article/view/1899 (besucht am 29. 11. 2020) (siehe S. 45).

[BM07] Philip A. Bernstein und Sergey Melnik. „Model Management 2.0: Manipulating Richer Mappings". In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. SIGMOD '07. New York, NY, USA: ACM, 2007, S. 1–12. ISBN: 978-1-59593-686-8. DOI: 10.1145/1247480.1247482. URL: http://doi.acm.org/10.1145/1247480.1247482 (besucht am 29. 11. 2020) (siehe S. 20).

[Bre00] Eric A. Brewer. *Towards Robust Distributed Systems*. Keynote. Portland, Oregon, USA, 2000. URL: https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf (besucht am 29. 11. 2020) (siehe S. 31).

[Bre11] Peter C. Breedveld. „Concept-Oriented Modeling of Dynamic Behavior". In: *Bond Graph Modelling of Engineering Systems: Theory, Applications and Software Support*. Hrsg. von Wolfgang Borutzky. New York, NY: Springer New York, 2011, S. 3–52. ISBN: 978-1-4419-9368-7. URL: http://dx.doi.org/10.1007/978-1-4419-9368-7_1 (besucht am 29. 11. 2020) (siehe S. 41).

[Bud09] Frank Budszuhn. *Subversion 1.5*. Galileo Press, 2009 (siehe S. 23, 33).

[Bür+14] Jens Bürger, Jan Jürjens, Thomas Ruhroth, Stefan Gärtner und Kurt Schneider. „Model-Based Security Engineering: Managed Co-evolution of Security Knowledge and Software Models". en. In: *Foundations of Security Analysis and Design VII*. Lecture Notes in Computer Science. Springer, Cham, 2014, S. 34–53. ISBN: 978-3-319-10081-4 978-3-319-10082-1. DOI: 10.1007/978-3-319-10082-1_2. URL: https://link.springer.com/chapter/10.1007/978-3-319-10082-1_2 (besucht am 29. 11. 2020) (siehe S. 51–53).

[Bus+14] Johannes Busse, Bernhard Humm, Christoph Lübbert, Frank Moelter, Anatol Reibold, Matthias Rewald, Veronika Schlüter, Bernhard Seiler, Erwin Tegtmeier und Thomas Zeh. „Was bedeutet eigentlich Ontologie?" de. In: *Informatik-Spektrum* 37.4 (Aug. 2014), S. 286–297. ISSN: 0170-6012, 1432-

122X. DOI: 10.1007/s00287-012-0619-2. URL: https://link.springer.com/article/10.1007/s00287-012-0619-2 (besucht am 29. 11. 2020) (siehe S. 14).

[Cam+16] Dario Campagna, Carlos Kavka, Alessandro Turco, Besian Pogace und Carlo Poloni. „Solving time-dependent coupled systems through FMI co-simulation and BPMN process orchestration". In: *2016 IEEE International Symposium on Systems Engineering (ISSE)*. Okt. 2016, S. 1–8. DOI: 10.1109/SysEng.2016.7753140 (siehe S. 41).

[Can+15] Lorenzo Canova, Simone Basso, Raimondo Iemma und Federico Morando. „Collaborative Open Data versioning: a pragmatic approach using Linked Data". In: *CeDEM15 - Conference for E-Democracy and Open Governement*. Krems, 2015, S. 171–183 (siehe S. 26, 55).

[CO05] John Christian und John Olds. „A Quantitative Methodology for Identifying Evolvable Space Systems". In: *1st Space Exploration Conference: Continuing the Voyage of Discovery*. Space Exploration Conferences. American Institute of Aeronautics und Astronautics, Jan. 2005. DOI: 10.2514/6.2005-2543. URL: https://arc.aiaa.org/doi/10.2514/6.2005-2543 (besucht am 29. 11. 2020) (siehe S. 18).

[DA10] Rim Djedidi und Marie-Aude Aufaure. „Ontology Evolution: State of the Art and Future Directions". en. In: *Ontology Theory, Management and Design: Advanced Tools and Models* (2010), S. 179–207. DOI: 10.4018/978-1-61520-859-3.ch008. URL: https://www.igi-global.com/chapter/ontology-evolution-state-art-future/42890 (besucht am 29. 11. 2020) (siehe S. 21, 22).

[DB07] Mark Dalgarno und Danilo Beuche. „Variant Management". In: *3rd British Computer Society Configuration Management Specialist Group Conference Variant Management*. 2007. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.132.1820 (besucht am 29. 11. 2020) (siehe S. 23).

[Dij59] Edsger W. Dijkstra. „A Note on Two Problems in Connexion with Graphs". In: *Numerische Mathematik* 1.1 (1959), S. 269–271 (siehe S. 75).

[DIN06] DIN EN ISO. *Ergonomie der Mensch-System-Interaktion – Teil 110: Grundsätze der Dialoggestaltung (ISO 9241-110:2006); Deutsche Fassung EN ISO 9241-110:2006*. Norm DIN EN ISO 9241-110. DIN EN ISO, Aug. 2006 (siehe S. 28).

[DIN14] DIN EN. *Speicherprogrammierbare Steuerungen Teil 3: Programmiersprachen*. Norm DIN EN 61131-3. DIN EN, Juni 2014 (siehe S. 15).

[DIN17] DIN EN. *Life-cycle-Management von Systemen und Produkten der Mess-, Steuer- und Regelungstechnik der Industrie (IEC 65/617/CDV:2016)*. Norm DIN EN 62890:2017-04;VDE 0810-890:2017-04 - Entwurf. DIN EN, Apr. 2017 (siehe S. 59).

[DIP11] Davide Di Ruscio, Ludovico Iovino und Alfonso Pierantonio. „What is Needed for Managing Co-evolution in MDE?“ In: *Proceedings of the 2Nd International Workshop on Model Comparison in Practice*. IWMCP '11. New York, NY, USA: ACM, 2011, S. 30–38. ISBN: 978-1-4503-0668-3. DOI: 10.1145/2000410.2000416. URL: http://doi.acm.org/10.1145/2000410.2000416 (besucht am 29. 11. 2020) (siehe S. 16, 17, 38).

[Dud] Dudenredaktion. *"Konsistenz" auf Duden online*. URL: http://www.duden.de/node/684770/revisions/1616335/view (besucht am 29. 11. 2020) (siehe S. 28).

[Eka+15] Fajar J. Ekaputra, Estefanía Serral, Marta Sabou und Stefan Biffl. „Knowledge Change Management and Analysis for Multi-Disciplinary Engineering Environments.“ In: *SEMANTiCS (Posters & Demos)*. 2015, S. 13–17. URL: https://pdfs.semanticscholar.org/2de2/75c1e716e54f5d3bfc48a8755415d20ccbcf.pdf (besucht am 29. 11. 2020) (siehe S. 26).

[Eng+01] Gregor Engels, Jochem M. Küster, Reiko Heckel und Luuk Groenewegen. „A Methodology for Specifying and Analyzing Consistency of Object-oriented Behavioral Models“. In: *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ESEC/FSE-9. New York, NY, USA: ACM, 2001, S. 186–195. ISBN: 978-1-58113-390-5. DOI: 10.1145/503209.503235. URL: http://doi.acm.org/10.1145/503209.503235 (besucht am 29. 11. 2020) (siehe S. 30).

[Fai+16] Sidra Faisal, Kemele M. Endris, Saeedeh Shekarpour, Sören Auer und Maria-Esther Vidal. „Co-evolution of RDF Datasets“. en. In: *Web Engineering*. Lecture Notes in Computer Science. Springer, Cham, Juni 2016, S. 225–243. ISBN: 978-3-319-38790-1 978-3-319-38791-8. DOI: 10.1007/978-3-319-38791-8_13. URL: https://link.springer.com/chapter/10.1007/978-3-319-38791-8_13 (besucht am 29. 11. 2020) (siehe S. 84, 86).

[Fen01] Dieter Fensel. *Ontologies:: A Silver Bullet for Knowledge Management and Electronic Commerce*. en. Berlin Heidelberg: Springer-Verlag, 2001. ISBN: 978-3-662-04396-7. URL: https://www.springer.com/de/book/9783662043967 (besucht am 29. 11. 2020) (siehe S. 14).

[FMP99] Pascal Fradet, Daniel Le Métayer und Michaël Périn. „Consistency Checking for Multiple View Software Architectures“. en. In: *Software Engineering — ESEC/FSE '99*. Springer, Berlin, Heidelberg, 1999, S. 410–428. DOI:

10.1007/3-540-48166-4_25. URL: https://link.springer.com/chapter/10.1007/3-540-48166-4_25 (besucht am 29. 11. 2020) (siehe S. 67).

[Fog05] Karl Fogel. *Producing open source software*. en. Safari Books Online. Sebastopol, Calif.: O'Reilly, 2005. ISBN: 978-0-596-00759-1 (siehe S. 23, 33).

[Fre06] Stefan Martin Frenz. „Zuverlässiger verteilter Speicher mit transaktionaler Konsistenz". de. Dissertation. Universität Ulm, Juni 2006. URL: https://oparu.uni-ulm.de/xmlui/handle/123456789/387 (besucht am 29. 11. 2020) (siehe S. 28).

[Fri+00] Ernst Fricke, Bernd Gebhard, Herbert Negele und Eduard Igenbergs. „Coping with changes: Causes, findings, and strategies". en. In: *Systems Engineering* 3.4 (2000), S. 169–179. ISSN: 1098-1241, 1520-6858. DOI: 10.1002/1520-6858(2000)3:4<169::AID-SYS1>3.0.CO;2-W. URL: http://doi.wiley.com/10.1002/1520-6858%282000%293%3A4%3C169%3A%3AAID-SYS1%3E3.0.CO%3B2-W (besucht am 29. 11. 2020) (siehe S. 9).

[Fro+16] Marvin Frommhold, Rubén Navarro Piris, Natanael Arndt, Sebastian Tramp, Niklas Petersen und Michael Martin. „Towards Versioning of Arbitrary RDF Data". In: *Proceedings of the 12th International Conference on Semantic Systems*. ACM, 2016, S. 33–40. URL: http://dl.acm.org/citation.cfm?id=2993327 (besucht am 29. 11. 2020) (siehe S. 26, 27).

[FS05] Ernst Fricke und Armin P. Schulz. „Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle". en. In: *Systems Engineering* 8.4 (Jan. 2005), S. 342–359. ISSN: 1520-6858. DOI: 10.1002/sys.20039. URL: http://onlinelibrary.wiley.com/doi/10.1002/sys.20039/abstract (besucht am 29. 11. 2020) (siehe S. 8–10).

[Fuh12] Insa Marie-Ann Fuhrmann. „Layout of Compound Graphs". Diploma Thesis. Kiel: Christian-Albrechts-Universität zu Kiel, Feb. 2012. URL: https://rtsys.informatik.uni-kiel.de/~biblio/downloads/theses/ima-dt.pdf (besucht am 29. 11. 2020) (siehe S. 128).

[Fuj99] Richard M. Fujimoto. *Parallel and Distribution Simulation Systems*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1999. ISBN: 0-471-18383-0 (siehe S. 4).

[Fun17] Jan Funke. „Integration von Co-Evolutionsstrategien in ein Revision Management System". Studienarbeit. Dresden: TU Dresden, Sep. 2017 (siehe S. 118, 138).

[Gal+15] Virginie Galtier, Stephane Vialle, Cherifa Dad, Jean-Philippe Tavella, Jean-Philippe Lam-Yee-Mui und Gilles Plessis. „FMI-based Distributed Multi-

simulation with DACCOSIM“. In: *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. DEVS '15. San Diego, CA, USA: Society for Computer Simulation International, 2015, S. 39–46. ISBN: 978-1-5108-0105-9. URL: http://dl.acm.org/citation.cfm?id=2872965.2872971 (besucht am 29. 11. 2020) (siehe S. 41).

[GHU14] Markus Graube, Stephan Hensel und Leon Urbas. „R43ples: Revisions for Triples-An Approach for Version Control in the Semantic Web.“ In: *LDQ@ SEMANTICS*. 2014 (siehe S. 2, 26, 27, 34, 51, 55, 106, 108).

[GHU16] Markus Graube, Stephan Hensel und Leon Urbas. „Open Semantic Revision Control with R43ples: Extending SPARQL to access revisions of Named Graphs“. In: *Proceedings of the 12th International Conference on Semantic Systems*. SEMANTiCS 2016. New York, NY, USA: ACM, 2016, S. 49–56. ISBN: 978-1-4503-4752-5. DOI: 10.1145/2993318.2993336. URL: http://doi.acm.org/10.1145/2993318.2993336 (besucht am 29. 11. 2020) (siehe S. 26, 27, 55, 110).

[GL02] Seth Gilbert und Nancy Lynch. „Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services“. In: *SIGACT News* 33.2 (Juni 2002), S. 51–59. ISSN: 0163-5700. DOI: 10.1145/564585.564601. URL: http://doi.acm.org/10.1145/564585.564601 (besucht am 29. 11. 2020) (siehe S. 31).

[Glo06] Klaus Gloede. *Skriptum zur Vorlesung Mathematische Logik*. Vorlesungsskript. Heidelberg: Mathematisches Institut der Universität Heidelberg, 2006. URL: http://math.uni-heidelberg.de/logic/md/lehre/mathlogik.pdf (besucht am 29. 11. 2020) (siehe S. 28).

[Gra16] Markus Graube. „Linked Enterprise Data als semantischer, integrierter Informationsraum für die industrielle Datenhaltung“. Dissertation. Dresden: TU Dresden, Nov. 2016 (siehe S. 2, 3, 10–12, 14, 103, 134).

[Hau+17] Christopher Haubeck, Alexander Pokahr, Winfried Lamersdorf, Abhishek Chakraborty, Jan Ladiges und Alexander Fay. „Evolution of cyber-physical production systems supported by community-enabled experiences“. In: *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*. Juli 2017, S. 867–874. DOI: 10.1109/INDIN.2017.8104886 (siehe S. 37).

[HBW15] Claudius Hauptmann, Michele Brocco und Wolfgang Wörndl. „Scalable Semantic Version Control for Linked Data Management.“ In: *LDQ@ ESWC*. 2015. URL: http://ceur-ws.org/Vol-1376/LDQ2015_paper_06.pdf (besucht am 29. 11. 2020) (siehe S. 27).

[Hen+16a] Stephan Hensel, Markus Graube, Leon Urbas, Till Heinzerling und Mathias Oppelt. „Co-Simulation mittels OPC UA“. In: *Tagungsband Automation 2016*. Baden-Baden, Juni 2016 (siehe S. 5, 44).

[Hen+16b] Stephan Hensel, Markus Graube, Leon Urbas, Till Heinzerling und Mathias Oppelt. „Co-simulation with OPC UA“. In: *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. Juli 2016, S. 20–25. DOI: 10.1109/INDIN.2016.7819127 (siehe S. 5, 15, 44).

[Hen+17] Stephan Hensel, Henry Bloch, Mario Hoernicke, Andreas Stutz, Christoph Kotsch, Thomas Holm, Jens Bernshausen, Simon Kronemeier, Axel Haller und Leon Urbas. „Beschreibung von Bedienbildern modularer Anlagen – Ergebnisse der NAMUR/ZVEI-Arbeitskreise (1.12.1 und 2.9.1) sowie des VDI/VDE-GMA FA 5.16“. In: *Tagungsband Automation 2017*. Baden-Baden, Juni 2017 (siehe S. 6).

[Hen13] Stephan Hensel. „Untersuchung von Synchronisierungsmechanismen in Linked Data Netzwerken“. Studienarbeit. Dresden: TU Dresden, Sep. 2013 (siehe S. 55, 103, 106).

[Hen14] Stephan Hensel. „Konflikterkennung und -behebung bei der Zusammenführung von revisionierten Graphen in Linked Data“. Diplomarbeit. Dresden: TU Dresden, Nov. 2014 (siehe S. 55, 103, 106, 113).

[Her05] Dietmar Hermsdörfer. *Generische Informationsmodellierung / semantische Brücke zwischen Daten und Diensten*. Heidelberg: Wichmann, 2005. ISBN: 978-3-87907-426-6 (siehe S. 11).

[HGU16] Stephan Hensel, Markus Graube und Leon Urbas. *Methodology for Conflict Detection and Resolution in Semantic Revision Control Systems*. Techn. Ber. 2016-08-A. Dresden: TU Dresden, Nov. 2016. URL: http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-211244 (besucht am 29. 11. 2020) (siehe S. 55, 86, 88–90, 113).

[HGU18] Stephan Hensel, Markus Graube und Leon Urbas. „Informationsmodelle im Lebenszyklus“. de. In: *atp edition* 60.4-5 (2018), S. 40–51. DOI: https://doi.org/10.17560/atp.v60i04-05.2345. URL: http://ojs.di-verlag.de/index.php/atp_edition/article/view/2345 (besucht am 29. 11. 2020) (siehe S. 44, 59, 60).

[HHH14] Bernhard Hoisl, Zhenjiang Hu und Soichiro Hidaka. „Towards co-evolution in model-driven development via bidirectional higher-order transformation“. In: *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. Jan. 2014, S. 466–471 (siehe S. 17, 38).

[Hil15] Frank Hilbert. „Kontextadaptive Informationsräume“. Dissertation. Dresden: TU Dresden, Nov. 2015. URL: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-198802 (besucht am 29. 11. 2020) (siehe S. 12).

[HKB17] Regina Hebig, Djamel E. Khelladi und Reda Bendraou. „Approaches to Co-Evolution of Metamodels and Models: A Survey“. In: *IEEE Transactions on Software Engineering* 43.5 (Mai 2017), S. 396–414. ISSN: 0098-5589. DOI: 10.1109/TSE.2016.2610424 (siehe S. 1, 59).

[Hoe+16] Mario Hoernicke, Christian Messinger, Esteban Arroyo und Alexander Fay. „Topologiemodelle in AutomationML“. In: *atp edition* 58.05 (Mai 2016), S. 28–41. ISSN: 2190-4111. URL: http://ojs.di-verlag.de/index.php/atp_edition/article/view/2300 (besucht am 29. 11. 2020) (siehe S. 45).

[Hof06] Douglas R. Hofstadter. *Gödel, Escher, Bach: ein endloses geflochtenes Band*. de. Klett-Cotta, 2006. ISBN: 978-3-608-94442-6 (siehe S. 28).

[Hof79] Douglas R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. New York, NY, USA: Basic Books, Inc., 1979. ISBN: 978-0-465-02685-2 (siehe S. 28).

[Hol+14] Thomas Holm, Michael Obst, Alexander Fay, Leon Urbas, Thomas Albers, Sven Kreft und Ulrich Hempen. „Dezentrale Intelligenz für modulare Automation“. In: *atp edition* 56.11 (Nov. 2014), S. 34–43. ISSN: 2190-4111. URL: ojs.di-verlag.de/index.php/atp_edition/article/view/2223 (besucht am 29. 11. 2020) (siehe S. 6).

[HP04] Jeff Heflin und Zhengxiang Pan. „A Model Theoretic Semantics for Ontology Versioning“. en. In: *The Semantic Web – ISWC 2004*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Nov. 2004, S. 62–76. ISBN: 978-3-540-23798-3 978-3-540-30475-3. DOI: 10.1007/978-3-540-30475-3_6. URL: https://link.springer.com/chapter/10.1007/978-3-540-30475-3_6 (besucht am 29. 11. 2020) (siehe S. 80).

[HR83] Theo Haerder und Andreas Reuter. „Principles of Transaction-oriented Database Recovery“. In: *ACM Comput. Surv.* 15.4 (Dez. 1983), S. 287–317. ISSN: 0360-0300. DOI: 10.1145/289.291. URL: http://doi.acm.org/10.1145/289.291 (besucht am 29. 11. 2020) (siehe S. 31).

[Huz+04] Zbigniew Huzar, Ludwik Kuzniarz, Gianna Reggio und Jean Louis Sourrouille. „Consistency Problems in UML-Based Software Development“. en. In: *UML Modeling Languages and Applications*. Springer, Berlin, Heidelberg, Okt. 2004, S. 1–12. DOI: 10.1007/978-3-540-31797-5_1. URL: https://link.springer.com/chapter/10.1007/978-3-540-31797-5_1 (besucht am 29. 11. 2020) (siehe S. 29, 30).

[HW14] Markus Herrmannsdörfer und Guido Wachsmuth. „Coupled Evolution of Software Metamodels and Models“. en. In: *Evolving Software Systems*. Hrsg. von Tom Mens, Alexander Serebrenik und Anthony Cleve. Springer Berlin Heidelberg, 2014, S. 33–63. ISBN: 978-3-642-45397-7 978-3-642-45398-4. DOI: 10.1007/978-3-642-45398-4_2. URL: http://link.springer.com/chapter/10.1007/978-3-642-45398-4_2 (besucht am 29. 11. 2020) (siehe S. 16).

[IEE98] IEEE. *IEEE Standard for Software Maintenance*. Norm IEEE Std 1219-1998. IEEE, 1998 (siehe S. 19).

[Int+19] Roberto Interdonato, Martin Atzmueller, Sabrina Gaito, Rushed Kanawati, Christine Largeron und Alessandra Sala. „Feature-rich networks: going beyond complex network topologies“. en. In: *Applied Network Science* 4.1 (Dez. 2019). ISSN: 2364-8228. DOI: 10.1007/s41109-019-0111-x. URL: https://appliednetsci.springeropen.com/articles/10.1007/s41109-019-0111-x (besucht am 29. 11. 2020) (siehe S. 15, 71).

[ISO01] ISO. *Software engineering - Product quality - Part 1: Quality model*. Norm ISO/IEC 9126-1:2001. ISO/IEC, Juni 2001 (siehe S. 19).

[ISO10] ISO IEC IEEE. *Systems and software engineering – Vocabulary*. Norm ISO/IEC/IEEE 24765:2010(E). ISO IEC IEEE, Dez. 2010, S. 1–418. URL: https://www.iso.org/standard/50518.html (besucht am 29. 11. 2020) (siehe S. 28).

[KB14] Uwe Kastens und Hans Kleine Büning. *Modellierung: Grundlagen und formale Methoden*. de. Carl Hanser Verlag GmbH Co KG, Okt. 2014. ISBN: 978-3-446-44249-8 (siehe S. 11).

[Keh+12] Timo Kehrer, Udo Kelter, Manuel Ohrndorf und Tim Sollbach. „Understanding model evolution through semantically lifting model differences with SiLift“. In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. Sep. 2012, S. 638–641. DOI: 10.1109/ICSM.2012.6405342 (siehe S. 37, 38).

[Keh15] Timo Kehrer. „Calculation and propagation of model changes based on user-level edit operations : a foundation for version and variant management in model-driven engineering“. Dissertation. Siegen: Universität Siegen, Okt. 2015. URL: https://nbn-resolving.org/urn:nbn:de:hbz:467-9633 (besucht am 29. 11. 2020) (siehe S. 3, 4, 35, 37, 51, 67, 81, 83, 86, 96, 99, 133).

[KF01] Michel Klein und Dieter Fensel. „Ontology Versioning on the Semantic Web“. In: *Proceedings of the First International Conference on Semantic Web Working*. SWWS'01. Aachen, Germany, Germany: CEUR-WS.org, 2001,

S. 75–91. URL: http://dl.acm.org/citation.cfm?id=2956602.2956610 (besucht am 29. 11. 2020) (siehe S. 26, 37, 38).

[Kle04] Michel Klein. „Change Management for Distributed Ontologies“. Dissertation. Vrije Universiteit, 2004 (siehe S. 82).

[Kol14] Lars Kolb. *NoSQL-Datenbanken Kapitel 1: Einführung*. Vorlesung. Universität Leipzig, 2014. URL: https://dbs.uni-leipzig.de/file/NoSQL_SS14_01_Intro.pdf (besucht am 29. 11. 2020) (siehe S. 31).

[Kön12] Barbara König. *Vorlesung Modellierung - Modellierungsmethoden der Informatik*. Vorlesung. Essen, 2012 (siehe S. 11).

[KVN12] Tommi Karhela, Antti Villberg und Hannu Niemistö. „Open ontology-based integration platform for modeling and simulation in engineering“. In: *International Journal of Modeling, Simulation, and Scientific Computing* 03.02 (Mai 2012), S. 1250004. ISSN: 1793-9623. DOI: 10.1142/S1793962312500043. URL: http://www.worldscientific.com/doi/abs/10.1142/S1793962312500043 (besucht am 29. 11. 2020) (siehe S. 44, 51, 53).

[Lad18] Jan Ladiges. „Automatisierte Bestimmung von Eigenschaften industrieller Produktionssysteme unter Einfluss evolutionärer Änderungen“. Dissertation. Hamburg: Helmut-Schmidt-Universität, 2018 (siehe S. 4, 16).

[Lee99] Y. Tina Lee. „Information Modeling: From Design to Implementation“. In: *Proceedings of the Second World Manufacturing Congress*. 1999, S. 315–321 (siehe S. 11, 12).

[Leh80] Meir M. Lehman. „Programs, life cycles, and laws of software evolution“. In: *Proceedings of the IEEE* 68.9 (1980), S. 1060–1076. ISSN: 0018-9219. DOI: 10.1109/PROC.1980.11805. URL: http://ieeexplore.ieee.org/document/1456074/ (besucht am 29. 11. 2020) (siehe S. 16, 20).

[Leh96] Meir M. Lehman. „Laws of software evolution revisited“. en. In: *Software Process Technology*. Springer, Berlin, Heidelberg, Okt. 1996, S. 108–124. DOI: 10.1007/BFb0017737. URL: https://link.springer.com/chapter/10.1007/BFb0017737 (besucht am 29. 11. 2020) (siehe S. 20).

[Lev+10] Tihamer Levendovszky, Bernhard Rumpe, Bernhard Schätz und Jonathan Sprinkle. „Model Evolution and Management“. In: *MBEERTS: Model-Based Engineering of Embedded Real-Time Systems*. Bd. LNCS 6100. arXiv: 1409.2361. Dagstuhl Castle: Springer Berlin Heidelberg, Okt. 2010, S. 241–270. URL: http://arxiv.org/abs/1409.2361 (besucht am 29. 11. 2020) (siehe S. 1–3, 15, 20, 35, 37, 39, 40).

[LFL16] Jan Ladiges, Alexander Fay und Winfried Lamersdorf. „Automated Determining of Manufacturing Properties and Their Evolutionary Changes

from Event Traces“. en. In: *Intelligent Industrial Systems* 2.2 (Juni 2016), S. 163–178. ISSN: 2199-854X. DOI: 10.1007/s40903-016-0048-7. URL: https://doi.org/10.1007/s40903-016-0048-7 (besucht am 29. 11. 2020) (siehe S. 4).

[LG11] Stefan Lier und Marcus Grünewald. „Net Present Value Analysis of Modular Chemical Production Plants“. en. In: *Chemical Engineering & Technology* 34.5 (Mai 2011), S. 809–816. ISSN: 1521-4125. DOI: 10.1002/ceat.201000380. URL: http://onlinelibrary.wiley.com/doi/10.1002/ceat.201000380/abstract (besucht am 29. 11. 2020) (siehe S. 5).

[LMT09] Francisco J. Lucas, Fernando Molina und Ambrosio Toval. „A systematic review of UML model consistency management“. In: *Information and Software Technology*. Quality of UML Models 51.12 (Dez. 2009), S. 1631–1645. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2009.04.009. URL: http://www.sciencedirect.com/science/article/pii/S0950584909000433 (besucht am 29. 11. 2020) (siehe S. 28–30, 39, 101).

[LR03] Meir M. Lehman und Juan F. Ramil. „Software evolution - Background, theory, practice“. In: *Information Processing Letters*. To honour Professor W.M. Turski's Contribution to Computing Science on the Occasion of his 65th Birthday 88.1 (Okt. 2003), S. 33–44. ISSN: 0020-0190. DOI: 10.1016/S0020-0190(03)00382-X. URL: http://www.sciencedirect.com/science/article/pii/S002001900300382X (besucht am 29. 11. 2020) (siehe S. 20).

[Lyo80] John Lyons. *Semantik*. Bd. 1. Beck'sche Elementarbücher. München: Beck, 1980. ISBN: 978-3-406-05272-9. URL: http://swbplus.bsz-bw.de/bsz010616691inh.htm (besucht am 29. 11. 2020) (siehe S. 11).

[Maj10] Frederic Majer. „Semantisches Informationsmodell für die Betriebsunterstützung dienstorientierter Systeme“. de. Dissertation. Karlsruhe: Karlsruher Institut für Technologie, 2010. URL: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000017356 (besucht am 29. 11. 2020) (siehe S. 11).

[Men17] Kim Mens. *Software Maintenance and Evolution*. Vorlesung im Kurs LINGI2252. Louvain-la-Neuve, Jan. 2017. URL: https://de.slideshare.net/kim.mens/software-maintenance-and-evolution (besucht am 29. 11. 2020) (siehe S. 19).

[Mod10] Modelisar. *Functional Mock-up Interface for Co-Simulation*. Techn. Ber. MODELISAR (07006) 1.0. MODELISAR consortium, Okt. 2010. URL: https://svn.modelica.org/fmi/branches/public/specifications/v1.0/FMI_for_CoSimulation_v1.0.pdf (besucht am 29. 11. 2020) (siehe S. 40).

[MS04] Bela Mutschler und Günther Specht. *Mobile Datenbanksysteme*. Xpert.press. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. ISBN: 978-3-642-62266-3 978-3-642-18731-5. DOI: 10.1007/978-3-642-18731-5. URL: http://link.springer.com/10.1007/978-3-642-18731-5 (besucht am 29. 11. 2020) (siehe S. 25, 26, 29).

[NAM13] NAMUR. *Anforderungen an die Automatisierungstechnik durch die Modularisierung verfahrenstechnischer Anlagen*. Namur Empfehlung NE 148. NAMUR, Okt. 2013 (siehe S. 1, 5).

[NFM17] Elisa Negri, Luca Fumagalli und Marco Macchi. „A Review of the Roles of Digital Twin in CPS-based Production Systems". In: *Procedia Manufacturing*. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy 11 (Jan. 2017), S. 939–948. ISSN: 2351-9789. DOI: 10.1016/j.promfg.2017.07.198. URL: http://www.sciencedirect.com/science/article/pii/S2351978917304067 (besucht am 29. 11. 2020) (siehe S. 1).

[NK04] Natalya F. Noy und Michel Klein. „Ontology Evolution: Not the Same as Schema Evolution". en. In: *Knowledge and Information Systems* 6.4 (Juli 2004), S. 428–440. ISSN: 0219-1377, 0219-3116. DOI: 10.1007/s10115-003-0137-2. URL: https://link.springer.com/article/10.1007/s10115-003-0137-2 (besucht am 29. 11. 2020) (siehe S. 2, 20, 21, 35, 40).

[Noy+06] Natalya F. Noy, Abhita Chugh, William Liu und Mark A. Musen. „A Framework for Ontology Evolution in Collaborative Environments". en. In: *The Semantic Web - ISWC 2006*. Springer, Berlin, Heidelberg, Nov. 2006, S. 544–558. DOI: 10.1007/11926078_39. URL: https://link.springer.com/chapter/10.1007/11926078_39 (besucht am 29. 11. 2020) (siehe S. 21, 36, 37, 51, 53, 54).

[OMG02] OMG. *Meta Object Facility (MOF) Specification*. Norm Version 1.4. OMG, Apr. 2002. URL: https://www.omg.org/spec/MOF/1.4/ (besucht am 29. 11. 2020) (siehe S. 13).

[ONF16] ONF. *Open Source SDN - NBI Information Model of Network Topology*. TR-1500 00. Open Networking Foundation, März 2016. URL: https://opennetworking.org/wp-content/uploads/2014/11/onf2014.314_NBI_Information_Models_-_Topology.13-2.pdf (besucht am 29. 11. 2020) (siehe S. 15).

[Opp+14] Mathias Oppelt, Gerrit Wolf, Oliver Drumm, Benjamin Lutz, Markus Stöß und Leon Urbas. „Automatic Model Generation for Virtual Commissioning based on Plant Engineering Data". In: *IFAC Proceedings Volumes*. 19th IFAC World Congress 47.3 (Jan. 2014), S. 11635–11640. ISSN: 1474-6670. DOI: 10.3182/20140824-6-ZA-1003.01512. URL: http://www.sciencedirect.

com/science/article/pii/S1474667016434671 (besucht am 29. 11. 2020) (siehe S. 4).

[OU14] Mathias Oppelt und Leon Urbas. „Integrated virtual commissioning an essential activity in the automation engineering process: From virtual commissioning to simulation supported engineering“. In: *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*. Okt. 2014, S. 2564–2570. DOI: 10.1109/IECON.2014.7048867 (siehe S. 4).

[OWU14] Mathias Oppelt, Gerrit Wolf und Leon Urbas. „Capability-analysis of co-simulation approaches for process industries“. In: *2014 IEEE Emerging Technology and Factory Automation (ETFA)*. Sep. 2014, S. 1–4. DOI: 10.1109/ETFA.2014.7005292 (siehe S. 4).

[Pap+13] Vicky Papavasileiou, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos und Vassilis Christophides. „High-level Change Detection in RDF(S) KBs“. In: *ACM Trans. Database Syst.* 38.1 (2013), 1:1–1:42. ISSN: 0362-5915. DOI: 10.1145/2445583.2445584. URL: http://doi.acm.org/10.1145/2445583.2445584 (besucht am 29. 11. 2020) (siehe S. 3, 4, 37, 39, 81–83, 86, 96, 111, 112, 133).

[Pha16] Tuyen Viet Pham. „Integration einer Benutzerverwaltung in das semantische Revisionsverwaltungssystem R43ples“. Studienarbeit. Dresden: TU Dresden, Sep. 2016 (siehe S. 118).

[Pie+18] Christopher Pietsch, Udo Kelter, Christopher Haubeck, Winfried Lamersdorf, Abhishek Chakraborty und Alexander Fay. „Using model differencing to reason about observable behavior changes of manufacturing systems“. In: *at - Automatisierungstechnik* 66.10 (2018), S. 795–805. ISSN: 0178-2312. DOI: 10.1515/auto-2018-0046. URL: https://www.degruyter.com/abstract/j/auto.2018.66.issue-10/auto-2018-0046/auto-2018-0046.xml (besucht am 29. 11. 2020) (siehe S. 37).

[RB06] Erhard Rahm und Philip A. Bernstein. „An Online Bibliography on Schema Evolution“. In: *SIGMOD Rec.* 35.4 (Dez. 2006), S. 30–31. ISSN: 0163-5808. DOI: 10.1145/1228268.1228273. URL: http://doi.acm.org/10.1145/1228268.1228273 (besucht am 29. 11. 2020) (siehe S. 20, 21).

[RB09] Matthias Riebisch und Stephan Bode. „Software-Evolvability“. de. In: *Informatik-Spektrum* 32.4 (Mai 2009), S. 339–343. ISSN: 0170-6012, 1432-122X. DOI: 10.1007/s00287-009-0349-2. URL: http://link.springer.com/article/10.1007/s00287-009-0349-2 (besucht am 29. 11. 2020) (siehe S. 1, 15, 16, 18, 19).

[RGU17] Julian Rahm, Markus Graube und Leon Urbas. „A roundtrip engineering approach for data consistency in process industry environments“. In: *2017*

IEEE 15th International Conference on Industrial Informatics (INDIN). Juli 2017, S. 559–564. DOI: 10.1109/INDIN.2017.8104833 (siehe S. 3).

[RLL98] David Rowe, John Leaney und David Lowe. „Defining systems evolvability - a taxonomy of change“. In: *International Conference and Workshop: Engineering of Computer-Based Systems*. Maale Hachamisha, Israel: IEEE Computer Society, Apr. 1998, S. 45–52. URL: http://www.researchgate.net/publication/232627101_Defining_Systems_Evolvability_-_A_Taxonomy_of_Change_(PDF) (besucht am 29. 11. 2020) (siehe S. 16, 18, 39).

[RR97] Young-Gook Ra und Elke A. Rundensteiner. „A transparent schema-evolution system based on object-oriented view technology“. In: *IEEE Transactions on Knowledge and Data Engineering* 9.4 (Juli 1997), S. 600–624. ISSN: 1041-4347. DOI: 10.1109/69.617053 (siehe S. 20).

[Ruh+14] Thomas Ruhroth, Stefan Gärtner, Jens Bürger, Jens Jürjens und Kurt Schneider. „Versioning and Evolution Requirements for Model-Based System Development“. In: *International Workshop on Comparison and Versioning of Software Models (CVSM 2014)*. Kiel, 2014 (siehe S. 16, 17, 35, 37–39, 51, 52).

[San05] Georg Sander. *Layout of compound directed graphs*. Techn. Ber. A/03/96. Universität des Saarlandes, Juni 2005. URL: http://scidok.sulb.uni-saarland.de/volltexte/2005/359/ (besucht am 29. 11. 2020) (siehe S. 68, 69).

[Sch+14] Nicole Schmidt, Arndt Lüder, Heinrich Steininger und Stefan Bifflˌ. „Analyzing requirements on software tools according to the functional engineering phase in the technical systems engineering process“. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. Sep. 2014, S. 1–8. DOI: 10.1109/ETFA.2014.7005144 (siehe S. 103).

[Sch16] Thomas Schmidt. *Verteilte Systeme - Replikation*. Vorlesung. Hamburg, 2016. URL: https://www.inet.haw-hamburg.de/teaching/ws-2016-17/verteilte-systeme/09_Replikation.pdf (besucht am 29. 11. 2020) (siehe S. 29).

[She97] A. Sheth. „Panel: Data Semantics: what, where and how?“ en. In: *Database Applications Semantics* (1997). Publisher: Springer, Boston, MA, S. 601–610. DOI: 10.1007/978-0-387-34913-8_26. URL: https://link.springer.com/chapter/10.1007/978-0-387-34913-8_26 (besucht am 29. 11. 2020) (siehe S. 11).

[Sie18] Siemens. *Mein Name ist Companion – Digital Companion*. de. Juli 2018. URL: https://new.siemens.com/global/de/unternehmen/stories/forschung-technologien/kuenstliche-intelligenz/kuenstliche-

intelligenz-digital-companion.html (besucht am 29. 11. 2020) (siehe S. 1).

[Sin11] Eric Sink. *Version Control by Example*. 2011. ISBN: 978-0-9835079-1-8. URL: http://ericsink.com/vcbe/vcbe_a4_lo.pdf (besucht am 29. 11. 2020) (siehe S. 24).

[SK09] Marc Shapiro und Bettina Kemme. „Eventual Consistency". en. In: *Encyclopedia of Database Systems*. Hrsg. von Ling Liu und M. Tamer Özsu. Springer US, 2009, S. 1071–1072. ISBN: 978-0-387-35544-3 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_1366. URL: http://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9_1366 (besucht am 29. 11. 2020) (siehe S. 31).

[Smo13] Peter Smolek. *Objektorientierte Modellierung und dynamische Co-Simulation mit CATIA V6 am Beispiel von Kraftfahrzeugsystemen*. Diplomarbeit. TU Wien, 2013. URL: http://publik.tuwien.ac.at/files/PubDat_224793.pdf (besucht am 29. 11. 2020) (siehe S. 4).

[SSS04] York Sure, Steffen Staab und Rudi Studer. „On-To-Knowledge Methodology (OTKM)". en. In: *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, Berlin, Heidelberg, 2004, S. 117–132. ISBN: 978-3-662-11957-0 978-3-540-24750-0. DOI: 10.1007/978-3-540-24750-0_6. URL: https://link.springer.com/chapter/10.1007/978-3-540-24750-0_6 (besucht am 29. 11. 2020) (siehe S. 59).

[Ste08] Perdita Stevens. „A Landscape of Bidirectional Model Transformations". en. In: *Generative and Transformational Techniques in Software Engineering II*. Hrsg. von Ralf Lämmel, Joost Visser und João Saraiva. Lecture Notes in Computer Science 5235. Springer Berlin Heidelberg, 2008, S. 408–424. ISBN: 978-3-540-88642-6 978-3-540-88643-3. DOI: 10.1007/978-3-540-88643-3_10. URL: http://link.springer.com/chapter/10.1007/978-3-540-88643-3_10 (besucht am 29. 11. 2020) (siehe S. 28).

[Sto04] Ljiljana Stojanovic. „Methods and tools for ontology evolution." Dissertation. Karlsruhe: Universitaet Fridericiana zu Karlsruhe, 2004 (siehe S. 3, 21, 35, 37, 51, 96, 99).

[Sur+08] Pradorn Sureephong, Nopasit Chakpitak, Yacine Ouzrout und Abdelaziz Bouras. „An Ontology-based Knowledge Management System for Industry Clusters". en. In: *Global Design to Gain a Competitive Edge*. Hrsg. von Xiu-Tian Yan, William J. Ion und Benoit Eynard. Springer London, 2008, S. 333–342. ISBN: 978-1-84800-239-5 (siehe S. 59).

[SUW18] Frank Stenger, Leon Urbas und Ljuba Woppowa. „100 % Digital in der Prozessindustrie". In: *CITplus* 21.10 (Okt. 2018), S. 6–8. ISSN: 1436-2597.

URL: https://www.chemanager-online.com/restricted-files/211803 (besucht am 29. 11. 2020) (siehe S. 1).

[SZ01] George Spanoudakis und Andrea Zisman. „Inconsistency management in software engineering: Survey and open research issues“. In: *Handbook of Software Engineering and Knowledge Engineering*. World Scientific, 2001, S. 329–380 (siehe S. 28).

[Tan06] Till Tantau. *Syntax versus Semantik Text und seine Bedeutung*. Vorlesung Logik für Informatiker. Lübeck, Okt. 2006. URL: https://caligari.dartmouth.edu/doc/texmf-dist/doc/latex/beamer/examples/a-lecture/beamerexample-lecture-beamer-version.pdf (besucht am 29. 11. 2020) (siehe S. 11).

[Tap99] Josef Tapken. „Implementing Hierarchical Graph-Structures“. en. In: *Fundamental Approaches to Software Engineering*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, März 1999, S. 219–233. ISBN: 978-3-540-65718-7 978-3-540-49020-3. DOI: 10.1007/978-3-540-49020-3_15. URL: https://link.springer.com/chapter/10.1007/978-3-540-49020-3_15 (besucht am 29. 11. 2020) (siehe S. 67).

[Tra+15] Sebastian Tramp, Ruben Navarro Piris, Timofey Ermilov, Niklas Petersen, Marvin Frommhold und Sören Auer. „Distributed linked data business communication networks: the LUCID endpoint“. In: *European Semantic Web Conference*. Springer, 2015, S. 154–158. URL: http://link.springer.com/chapter/10.1007/978-3-319-25639-9_30 (besucht am 29. 11. 2020) (siehe S. 27).

[Van+13] Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Sam Coppens, Erik Mannens und Rik Van de Walle. „R&Wbase: git for triples.“ In: *Proceedings of the 6th Workshop on Linked Data on the Web*. Mai 2013. URL: http://ceur-ws.org/Vol-996/papers/ldow2013-paper-01.pdf (besucht am 29. 11. 2020) (siehe S. 26, 27, 55).

[Van+15] Bert Van Acker, Joachim Denil, Hans Vangheluwe und Paul De Meulenaere. „Generation of an Optimised Master Algorithm for FMI Co-simulation“. In: *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. DEVS '15. San Diego, CA, USA: Society for Computer Simulation International, 2015, S. 205–212. ISBN: 978-1-5108-0105-9. URL: http://dl.acm.org/citation.cfm?id=2872965.2872993 (besucht am 29. 11. 2020) (siehe S. 41, 129).

[Van91] Johan Vanslembrouck. „A connection-oriented model for service description“. In: *Global Telecommunications Conference, 1991. GLOBECOM '91. 'Countdown to the New Millennium. Featuring a Mini-Theme on: Personal*

Communications Services. Dez. 1991, 802–806 vol.2. DOI: 10.1109/GLOCOM.1991.188493 (siehe S. 15).

[VDI16] VDI VDE. *Middleware in der Automatisierungstechnik - Vorgehensmodell für den Middleware Engineering Prozess*. Norm VDI/VDE 2657-2. VDI, 2016 (siehe S. 13).

[VDI17] VDI VDE NAMUR. *Automatisierungstechnisches Engineering modularer Anlagen in der Prozessindustrie – Allgemeines Konzept und Schnittstellen*. Norm VDI/VDE/NAMUR-Richtlinie 2658 - Blatt 1 (Gründruck). VDI VDE NAMUR, 2017 (siehe S. 44, 45).

[VDI18] VDI VDE NAMUR. *Automatisierungstechnisches Engineering modularer Anlagen in der Prozessindustrie – Modellierung von Bedienbildern*. Norm VDI/VDE/NAMUR-Richtlinie 2658 - Blatt 2 (Gründruck). VDI VDE NAMUR, 2018 (siehe S. 15, 46).

[VDI19] VDI. *Verfahrenstechnische Anlagen, Modulare Anlagen, Grundlagen*. Norm VDI 2776 - Blatt 1 (Entwurf). VDI, 2019 (siehe S. 1).

[VG06] Max Völkel und Tudor Groza. „SemVersion: An RDF-based Ontology Versioning System“. In: *Proceedings of IADIS International Conference on WWW/Internet*. Bd. 1. Murcia, Spain: IADIS, 2006, S. 195–202 (siehe S. 26).

[Vog+15a] Birgit Vogel-Heuser, Alexander Fay, Ina Schaefer und Matthias Tichy. „Evolution of software in automated production systems: Challenges and research directions“. In: *Journal of Systems and Software* 110 (Dez. 2015), S. 54–84. ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.08.026. URL: http://www.sciencedirect.com/science/article/pii/S0164121215001818 (besucht am 29. 11. 2020) (siehe S. 20).

[Vog+15b] Birgit Vogel-Heuser, Stefan Feldmann, Jens Folmer, Jan Ladiges, Alexander Fay, Sascha Lity, Matthias Tichy, Matthias Kowal, Ina Schaefer, Christopher Haubeck, Winfried Lamersdorf, Timo Kehrer, Sinem Getir, Mattias Ulbrich, Vladimir Klebanov und Bernhard Beckert. „Selected challenges of software evolution for automated production systems“. In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. Juli 2015, S. 314–321. DOI: 10.1109/INDIN.2015.7281753 (siehe S. 3, 15, 20, 33).

[Vos09] Gottfried Vossen. „ACID Properties“. en. In: *Encyclopedia of Database Systems*. Hrsg. von Ling Liu und M. Tamer Özsu. Springer US, 2009, S. 19–21. ISBN: 978-0-387-35544-3 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_831. URL: http://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9_831 (besucht am 29. 11. 2020) (siehe S. 31).

[VR15] Birgit Vogel-Heuser und Susanne Rösch. „Applicability of Technical Debt as a Concept to Understand Obstacles for Evolution of Automated Production Systems“. In: *2015 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Okt. 2015, S. 127–132. DOI: 10.1109/SMC.2015.35 (siehe S. 20).

[W3C] W3C. *W3C Data Activity - Building the Web of Data*. URL: https://www.w3.org/2013/data/ (besucht am 29. 11. 2020) (siehe S. 14).

[W3C12] W3C. *GLD Life cycle*. Aug. 2012. URL: https://www.w3.org/2011/gld/wiki/GLD_Life_cycle (besucht am 29. 11. 2020) (siehe S. 59).

[Wal10] Krzysztof Walkowiak. „Anycasting in connection-oriented computer networks: Models, algorithms and results“. en. In: *International Journal of Applied Mathematics and Computer Science* 20.1 (2010), S. 207–220. ISSN: 1641-876X. DOI: 10.2478/v10006-010-0015-5. URL: http://pldml.icm.edu.pl/pldml/element/bwmeta1.element.bwnjournal-article-amcv20i1p207bwm (besucht am 29. 11. 2020) (siehe S. 15).

[Wes+01] Andrea Westerinen, John Schnizlein, John Strassner, Mark Scherling, Bob Quinn, Shai Herzog, An-Ni Huynh und Mark Carlson. *Terminology for Policy-Based Management*. 2001. URL: https://tools.ietf.org/html/rfc3198 (besucht am 29. 11. 2020) (siehe S. 11, 12).

[Yan15] Xinyu Yang. „Erweiterte Merging-Funktionalitäten für semantische Revisionsverwaltungssysteme“. Studienarbeit. Dresden: TU Dresden, Aug. 2015 (siehe S. 118).

Ingenieure wollen immer alles ganz genau wissen. Wie wär's mit einem E-Paper- oder Zeitungs-Abo?

Mehr Meinung. Mehr Orientierung. Mehr Wissen.
Wesentliche Informationen zu neuen Technologien und Märkten.
Das bietet VDI nachrichten, Deutschlands meinungsbildende Wochenzeitung zu Technik, Wirtschaft und Gesellschaft, den Ingenieuren. Sofort abonnieren und lesen.
Donnerstagabends als E-Paper oder freitags als Zeitung.

Jetzt abonnieren: Leser-Service VDI nachrichten, 65341 Eltville
Telefon: +49 6123 9238-201, Telefax: +49 6123 9238-244, vdi-nachrichten@vuservice.de

100 JAHRE VDI nachrichten
TECHNIK IN SZENE GESETZT.

www.vdi-nachrichten.com/abo

Die Reihen der Fortschritt-Berichte VDI:

1 Konstruktionstechnik/Maschinenelemente
2 Fertigungstechnik
3 Verfahrenstechnik
4 Bauingenieurwesen
5 Grund- und Werkstoffe/Kunststoffe
6 Energietechnik
7 Strömungstechnik
8 Mess-, Steuerungs- und Regelungstechnik
9 Elektronik/Mikro- und Nanotechnik
10 Informatik/Kommunikation
11 Schwingungstechnik
12 Verkehrstechnik/Fahrzeugtechnik
13 Fördertechnik/Logistik
14 Landtechnik/Lebensmitteltechnik
15 Umwelttechnik
16 Technik und Wirtschaft
17 Biotechnik/Medizintechnik
18 Mechanik/Bruchmechanik
19 Wärmetechnik/Kältetechnik
20 Rechnerunterstützte Verfahren (CAD, CAM, CAE CAQ, CIM ...)
21 Elektrotechnik
22 Mensch-Maschine-Systeme
23 Technische Gebäudeausrüstung

ISBN 978-3-18-387310-4