Fortschritt-Berichte VDI

VDI

Reihe 8

Mess-,
Steuerungs- und
Regelungstechnik

Nr. 1266

M.Sc. Constantin Wagner,
Karben

Konzept zur Unterstützung der dezentralen Wiederverwendung in komponentenbasierten Systemen der operativen Leittechnik

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

“Konzept zur Unterstützung der dezentralen Wiederverwendung in komponentenbasierten Systemen der operativen Leittechnik“

Von der Fakultät für Georessourcen und Materialtechnik
der Rheinisch-Westfälischen Technischen Hochschule Aachen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigte Dissertation

vorgelegt von **M. Sc.**

Constantin August Wilhelm Wagner

aus Berlin.

Berichter: Univ.-Prof. Dr.-Ing. Ulrich Epple
Univ.-Prof. Dr.-Ing. Alexander Fay

Tag der mündlichen Prüfung: 17. Mai 2019

Fortschritt-Berichte VDI

Reihe 8

Mess-, Steuerungs- und Regelungstechnik

M.Sc. Constantin Wagner, Karben

Nr. 1266

Konzept zur Unterstützung der dezentralen Wiederverwendung in komponentenbasierten Systemen der operativen Leittechnik

Lehrstuhl für Prozessleittechnik der RWTH Aachen

Wagner, Constantin
Konzept zur Unterstützung der dezentralen Wiederverwendung in komponentenbasierten Systemen der operativen Leittechnik
Fortschr.-Ber. VDI Reihe 08 Nr. 1266. Düsseldorf: VDI Verlag 2019.
160 Seiten, 54 Bilder, 3 Tabellen.
ISBN 978-3-18-526608-9 ISSN 0178-9546,
€ 57,00/VDI-Mitgliederpreis € 51,30.

Für die Dokumentation: Prozessleittechnik – Wiederverwendung – Variantenmodell – Komponentensysteme – Delta-Modellierung

Die Wiederverwendung von Teillösungen in komponentenbasierten Systemen der Automatisierungstechnik findet in der Praxis, wenn überhaupt, wenig systematisch statt. In dieser Arbeit wird ein Konzept zur Beschreibung und Wiederverwendung von Komponentensystemen vorgestellt. Grundlagen des Konzepts sind eine auf Delta-Modellen basierende Variantenbeschreibung und ein Modell zur Abstraktion von Komponentensystemen. Dadurch ist es möglich, Komponentensysteme unabhängig von deren konkreter Realisierung zu beschreiben und wiederzuverwenden. Für die Anwendung in der Praxis wird zusätzlich ein Mechanismus zur Bekanntmachung und Verteilung der Wiederverwendungsgegenstände vorgestellt. Dieser besteht aus Prozessen zur Anwendung der Modelle sowie der dafür notwendigen Dienste. Das Konzept wurde prototypisch realisiert und an Software- und hybriden Systemen erprobt.

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.

D82 (Diss. RWTH Aachen University, 2019)
Tag der mündlichen Prüfung: 17. Mai 2019

ISSN 0178-9546
ISBN 978-3-18-526608-9

Vorwort

Die vorliegende Arbeit entstand während meiner Zeit als Mitarbeiter am Lehrstuhl für Prozessleittechnik der RWTH Aachen. Auf diese Jahre blicke ich mit Dankbarkeit für die Unterstützung und die gewährten Chancen zurück. Ich möchte mich an dieser Stelle bei allen bedanken, die mir während dieser Zeit zur Seite gestanden haben.

Mein besonderer Dank gilt Herrn Professor Dr.-Ing. Ulrich Epple für die Unterstützung meines Promotionsvorhabens. Die konstruktive Atmosphäre am Lehrstuhl und der ausgezeichnete fachliche Austausch mit ihm bildeten die Basis für den erfolgreichen Abschluss meiner Arbeit. Durch die gebotenen Freiräume konnte ich unterschiedliche Facetten der Automatisierungstechnik kennenlernen und so meinen Horizont erweitern.

Bei Herrn Professor Dr.-Ing. Alexander Fay, Inhaber der Professur für Automatisierungstechnik der Helmut-Schmidt-Universität / Universität der Bundeswehr, möchte ich mich für die Übernahme der Rolle des Zweitgutachters und lehrreichen Gespräche bedanken.

Ich danke meinen Kollegen für die gute Zusammenarbeit und die interessanten, teils auch kontroversen, Diskussionen. Bei Frau Milescu bedanke ich mich insbesondere für die organisatorische Hilfe und die gute Zusammenarbeit bei der Lehrstuhlverwaltung.

Bei den Mitgliedern des Arbeitskreises 2.2 „Prozessführung“ der NAMUR bedanke ich mich für den lehrreichen fachlichen Austausch. Durch die Gespräche habe ich wertvolle Einblicke in die Praxis erhalten.

Bei meiner Freundin Katharina Schüller möchte ich mich für die Unterstützung während der vergangenen Jahre bedanken. Insbesondere während der intensiven Phasen war sie mir ein großer Rückhalt.

Abschließend danke ich meinen Eltern Albertine und Michael Wagner sowie meiner Schwester Charlotte für die Unterstützung während meines Studiums und anschließend während meiner Promotion. Immer standen sie mir mit Rat und Tat zur Seite.

Karben, im Mai 2019

Constantin Wagner

Inhaltsverzeichnis

Abkürzungen

DDS Data Distribution Service

ERP Enterprise Resource Planning

I4.0 Industrie 4.0

MES Manufacturing Execution System

MTP Module Type Package

MPC Model Predictive Control

NAMUR Interessengemeinschaft Automatisierungstechnik der Prozessindustrie

OMG Object Management Group

OPC UA OPC Unified Architecture

PLS Prozessleitsystem

POE Programmorganisationseinheit

SOA Service orientierte Architektur

SPS Speicherprogrammierbare Steuerung

UML Unified Modeling Language

Kurzfassung

Im Rahmen der Entwicklung von Automatisierungslösungen sind der überwiegende Anteil der durchgeführten Arbeiten repetitiver Art. Diese wiederholenden Arbeiten sind fehleranfällig und entscheiden nur in geringem Maße über den Projekterfolg. Daher lohnt es sich, die Mitarbeiter bei diesen Arbeiten zu unterstützen, so dass mehr Ressourcen für andere Tätigkeiten zur Verfügung stehen. Die entwickelten Automatisierungslösungen ähneln sich sehr häufig funktional oder auch ihre Implementierung betreffend. Dies führt zur Wiederverwendung von Lösungen bzw. Teillösungen. Diese Wiederverwendung ist meist unsystematisch und nicht explizit dokumentiert. Daraus ergeben sich Herausforderungen für die Erweiterung oder Änderung von mehrfach genutzten Lösungen. Im Zuge des demografischen Wandels und der damit einhergehenden Verknappung von qualifizierten Arbeitskräften muss die Arbeit effizienter werden. Eine Möglichkeit die Effizienz zu verbessern ist die Nutzung von Methoden und Tools zur Unterstützung der Wiederverwendung.

In der Automatisierungstechnik ist der Einsatz von komponentenbasierten Systemen (z. B. Funktionsbausteine IEC 61131 oder Package Units) sehr verbreitet. Dabei handelt es sich um hybride Systeme, d.h. sowohl um Hard- als auch um Softwaresysteme. Der Fokus der Automatisierungstechnik liegt auf der Betrachtung von hybriden Systemen.

In der vorliegenden Arbeit wird ein Konzept zur Unterstützung der Wiederverwendung in komponentenbasierten Architekturen vorgestellt. Als erstes wird dafür der Begriff der Komponente sowohl für Hard- als auch für Softwarekomponenten gleichermaßen definiert. Damit ist es möglich, die unterschiedlichen Komponentenarten gleich zu behandeln. Ergebnis dieser Betrachtung ist ein Metamodell für die Beschreibung von Komponentensystemen. Das Komponenten-Modell besteht aus einem Teil zur Beschreibung von Komponententypen und einem zweiten Teil zur Beschreibung von Systemen, die aus Instanzen zusammengesetzt sind. Kern des Konzepts zur Unterstützung der Wiederverwendung ist das Delta-Modell aus der Informatik. Dieser Ansatz beschreibt die Variabilität im Problemraum. So können implementierungsspezifische Unterschiede zwischen Varianten modelliert werden. Für die Beschreibung der Delta-Modelle wird ein objektorientiertes Delta-Modell vorgestellt. Auf Basis des Delta-Modells wird ein Abstandsmaß für die Beschreibung der Unterschiedlichkeit von Varianten vorgestellt.

Aufbauend auf dem Komponenten- und dem Delta-Modell wird ein Mechanismus zur Nutzung der beschriebenen Varianten in der dezentralen Entwicklung und Verwendung von Automatisierungslösungen vorgestellt. Grundlage für den Mechanismus ist eine Server-Client Architektur. Auf dem Server werden alle für den Kontext relevanten Typ- und Delta-Modelle gespeichert. Aus diesen können für den konkreten Anwendungsfall die Instanz-Modelle gebildet und auf den Client heruntergeladen werden. Mit den auf dem Client vorhandenen Typ-Modellen, den darin enthaltenen Referenzen zwischen Komponenten-Typ-Modellen und ihren Implementierungen können die konkreten Komponentensysteme

aufgebaut werden. Der Manager des Clients führt eine Liste der verwendeten Varianten und bietet die Möglichkeit, nach neueren Varianten auf dem Server zu suchen und diese herunterzuladen.

Das Konzept unterstützt den Nutzer bei der Wiederverwendung in komponentenbasierten Systemen. Es bietet mit dem Delta-Modell einen Mechanismus zur abstrakten Beschreibung des Wiederverwendungsgegenstands. Durch das Komponenten-Modell können sowohl die Komponenten-Typen als auch die Systeme aus Instanzen unabhängig von ihrer konkreten Implementierung beschrieben werden. Durch diese Trennung zwischen dem konkreten System und der Variabilitätsbeschreibung wird das in den Komponentensystemen enthaltene Wissen leicht in neue Systeme übertragbar. Dazu müssen die Typ-Modelle auf die jeweiligen Implementierungen projiziert werden. Zusätzlich wird eine Trennung zwischen den Versionen einer Komponente und den Varianten eines Komponentensystems geschaffen.

Für die Wiederverwendung von Lösungen ist es erforderlich, dass der potentielle Nutzer von einer bereits bestehenden Lösung Kenntnis hat. Zur Förderung der Bekanntheit von bestehenden Lösungen und zur Unterstützung von deren Austausch zwischen verschiedenen Systemen dient der Mechanismus zur dezentralen Nutzung. Nutzer können so die vorhandenen Lösungen erkunden und die Zusammenhänge zwischen ihnen erkennen.

Das vorgestellte Konzept wurde in am Lehrstuhl für Prozessleittechnik der RWTH Aachen entwickelten Laufzeitumgebung ACPLT/RTE prototypisch umgesetzt und erprobt. Die beschriebenen Modelle wurden in Bibliotheken der Laufzeitumgebung realisiert. Die Architektur mit den Managern für Server und Client, sowie die notwendigen Transformationen der Modelle, wurden ebenso implementiert. An drei Anwendungsfällen wurde das Konzept erprobt. Erster Anwendungsfall ist die Beschreibung von Bausteinen für PID-Regler. Die verschiedenen Ausprägungen der Prozessführungskomponenten des Lehrstuhls sind der zweite Anwendungsfall. Als Testfall für ein hybrides System diente die modulare Anlage des Lehrstuhls für Prozessleittechnik.

Abstract

A huge part of the tasks conducted in the development of automation solutions is repetitive. Those repetitive work is error-prone and mostly not that relevant for the success of projects. Hence, it is beneficial to support the user to fulfill these repetitive tasks such that more resources can be used for tasks that are more relevant for the project's success. Most of the developed automation solutions are similar regarding their functionality or their implementation. This leads to the reuse of solutions or solution parts. Nowadays, this reuse is mostly unsystematic and not documented explicitly. As a result, extending or modifying solutions that are used more than once is challenging. Considering the demographic change and the resulting reduction of manpower, working must be more efficient. One way to increase the efficiency is to use methods and tools to support the reuse of solutions in an appropriate way.

Component-based systems are very much used in the automation domain. Examples are function blocks of the IEC 61131 or package units. Regarding the focus of the automation domain, most systems are hybrid systems. Those systems can consist of hard and software components.

In this thesis, a concept to support the reuse in component-based architectures is introduced. First, the definition for components used in this thesis are presented. These cover both hard and software components. Starting from the definition a metamodel for the description of component systems. It consists of a part for describing component types and another part to describe the systems built of the instances. Core element of the concept to support the reusability is the delta model that is well-known in the software development domain. Those delta models are used to describe the variability in the solution space. The great advantage of these models is that implementation specific differences between variants can be considered. The delta models are described by an object-oriented model and are the basis for an approach to calculate the differences between variants.

Additionally to the component and delta model, a mechanism to use the variants of automation solutions in a decentralized development and usage is presented. Basis for this mechanism is a server- client architecture. Inside the server, all relevant type and delta models are stored. By combining these models, the concrete instance models describing a specific variant can be generated and sent to a client. By mapping between the types and corresponding components on the client, the instance model can be transformed in a component model. The client's manager holds a list of all variants used on the client and provides the feature to search for new variants on the server. Found variants can be downloaded from the server by the client manager.

This concept supports users with the reuse in component-based systems. The delta model is provided as a mechanism for the abstract description of the subject of reusability. The component model allows it to handle solutions in an abstract manner, independent from

their concrete realization. By this separation, the knowledge included in component-based systems can easily be transferred into new systems. Only the transformation of types into new implementations is required. Additionally, the concept supports the separation between variants and versions. Different realizations of the same component type are versions of the component. Because those realizations are compatible to each other their change has no effect on a component system. Change in the structure or the types of components in a component system leads to a new variant.

In order to reuse solutions, it is necessary that a potential user is aware of an existing solution. To support the prominence of existing solutions and their exchange between different systems the mechanism for the decentralized usage is introduced. Users can explore existing solutions and the relations between them.

The concept was prototypically implemented in the runtime environment of the Chair of Process Control Engineering at RWTH Aachen University. The introduced models have been implemented in the object-oriented structure of the runtime environment. The decentralized approach along with the managers for the server and client have been realized as well. Three use cases have been used to test the concept. The first one is modeling and handling the different variants of a PID control block. The description of the various types of process control blocks of the Chair of Process Control Engineering is the second use case. As the third testcase for hybrid systems the chairs modular plant was utilized.

1 Einführung

Die vorliegende Arbeit stellt ein Konzept zur Unterstützung der dezentralen Wiederverwendung in komponentenbasierten Systemen der operativen Leittechnik vor. Dazu werden in diesem Kapitel zunächst die Motivation und der Aufbau der Arbeit erläutert. Zunächst wird erklärt, warum es notwendig ist, sich mit der Wiederverwendung in der Automatisierungstechnik zu befassen. Danach erfolgt die Vorstellung der konkreten Problemstellung und des gewählten Lösungswegs. Abschließend wird die Struktur der Arbeit erklärt.

1.1 Motivation

Aktuell besteht die Herausforderung der Automatisierung nicht in der Herstellung von Produkten, sondern darin, die dafür nötigen Anlagen wirtschaftlich zu betreiben und die Produktion an die Marktanforderungen anzupassen [SAG+17]. Die Herstellung der Produkte ist technisch möglich. Allerdings stellt die wirtschaftliche Produktion von kleinen Stückzahlen und der Wechsel zwischen Produkten in kurzen Zeiträumen eine große Herausforderung dar. Im Folgenden werden die Motive für die Beschäftigung mit Wiederverwendung aufgezählt und anschließend genauer beschrieben:

1. Die Automatisierungstechnik wird durch neue Anforderungen zunehmend komplexer. Dies sind beispielsweise die Produktion kleiner Stückzahlen oder die Sicherstellung einer höheren Anlagenverfügbarkeit.
2. Ein hoher Anteil der Tätigkeiten im Engineering ist repetitiv. Diese Arbeiten sind nötig für die Umsetzung der Projekte, bestimmen allerdings nicht den Projekterfolg.
3. Der steigende Kostendruck in der Produktion führt zu einem ebenso steigenden Kostendruck in der Automatisierungstechnik. Dadurch müssen die Aufwände für die Entwicklung und Wartung der Automatisierungslösungen reduziert werden.
4. Ähnliche, d. h. sich leicht unterscheidende Problemstellungen, führen zu Lösungen für Aufgaben in der Automatisierungstechnik, die sich nur marginal voneinander unterscheiden.
5. Komponentenorientierte Architekturen, z. B. nach der IEC 61131 [IEC14b], werden auch in einer von Servicearchitekturen geprägten Automatisierungstechnik die Basis für die Entwicklung von Lösungen bilden.
6. Die großen Hindernisse bei der Wiederverwendung von Lösungen sind weniger technischer, sondern vielmehr organisatorischer Natur. Zudem findet in der Prozessindustrie eine systematische Wiederverwendung bestehender Komponentensysteme in den meisten Fällen nicht statt.

Zunehmnde Komplexität in der Automatisierungstechnik
Die Automatisierungstechnik hat die Aufgabe, Maschinen in die Lage zu versetzen, (Produktions-) Prozesse teilweise oder vollständig autonom durchführen zu können. Antrieb für den Einsatz von Automatisierung können verschiedene Gründe sein, z. B., dass ein Prozess nicht von Menschen durchgeführt werden kann oder dass die Durchführung nicht ökonomisch sinnvoll ist. Zur Realisierung der gestellten Aufgaben greifen Automatisierungstechniker neben den Methoden der Regelungstechnik auf Konzepte und Technologien aus anderen Domänen zurück. Der klassische Aufbau der Automatisierungstechnik wird in Form einer geschichteten Pyramide beschrieben (Abbildung 1.1). Im unteren Bereich ist das Feld, d. h. die Maschinen und Anlagen, die den (Produktions-) Prozess umsetzen, zu erkennen. In den darüber liegenden Schichten werden die Automatisierungsfunktionen realisiert. Zwischen den Ebenen werden Prozess- und Planungsinformationen ausgetauscht. Die Planungsinformationen fließen von oben nach unten und die Prozessinformationen von unten nach oben, wobei keine Ebene übersprungen wird. Jede Ebene aggregiert die Informationen bzw. Funktionalitäten der Ebenen darunter. Dies wird durch die spitz zulaufende Pyramide verdeutlicht.

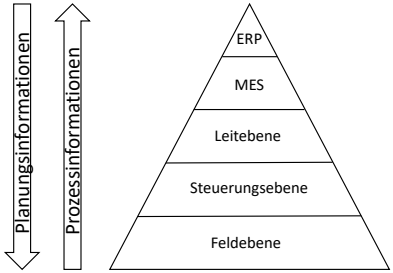


Abbildung 1.1: Automatisierungspyramide nach [SEE09]

Der strikt in Ebenen unterteilte Aufbau hat sich in der Vergangenheit bei der Entwicklung von sicheren und zuverlässigen Lösungen von Aufgaben der Automatisierungstechnik bewährt. Durch Industrie 4.0 (I4.0) wird die industrielle Produktion weiterentwickelt, indem Geräte oder Softwaresysteme miteinander interagieren, auch wenn diese nicht derselben oder benachbarten Ebenen angehören. Zusätzlich dazu treiben moderne Kommunikationsprotokolle wie OPC Unified Architecture (OPC UA) [PGP+15] und Data Distribution Service (DDS) [AE17] die Entwicklung der industriellen Produktion voran. Dies führt zu einer Flexibilisierung der Automatisierungstechnik und der industriellen Produktion [SAG+17]. Dadurch wird eine wirtschaftliche Produktion von kleinen Stückzahlen bis hin zur Losgröße eins möglich.

Die stärkere Vernetzung kann durch ein Aufbrechen der festen Strukturen der Pyramide zugunsten einer flexibleren Servicearchitektur [EE13, WE17] entstehen. Alternativ kann die Servicearchitektur zum bestehenden Ausbau hinzugefügt werden [LU17]. Dadurch wird die Hard- und Software der Automatisierungstechnik insgesamt komplexer und der Aufwand, sie zu entwickeln und zu betreiben, steigt. Analog dazu werden die Aufwände für das Engineering von Automatisierungslösungen und die Integration der einzelnen Anwendungen in ein Gesamtsystem (Systemengineering) [DMG+17] größer.

Hoher Anteil repetitiver Tätigkeiten im Engineering
Bei der Mehrzahl der Tätigkeiten im Engineering, d. h. in der Erstellung von Lösungen von Automatisierungsaufgaben, handelt es sich um repetitive Tätigkeiten [Web14]. Lediglich ein kleiner Anteil der Tätigkeiten ist für den Erfolg oder Misserfolg von Projekten verantwortlich. Die Bearbeiter müssen sich daher auf diese Tätigkeiten konzentrieren und dafür von den repetitiven Tätigkeiten entlastet werden. Das folgende Zitat von [Web14] verdeutlicht den beschriebenen Sachverhalt:

> Ungefähr 90 Prozent der Engineeringleistungen sind normale, wiederkehrende Ingenieurtätigkeiten. Die restlichen 10 Prozent Engineeringleistungen aber sind das Besondere, das Wichtige. Sie sind der Erfahrungsschatz oder das Innovative und nicht selten auch das Firmen-Know-how. Von der gewissenhaften Umsetzung dieser 10 Prozent Planungsvorgaben hängt aber i. d. R. wesentlich der Projekterfolg ab. [Web14]

Steigender Kostendruck
Im Rahmen der angesprochenen Flexibilisierung der Produktion rücken nicht-funktionale Anforderungen in den Fokus [BUN, Lee08]. Beispiele dafür sind die Reduktion von Kosten für die Erstellung und Wartung einer Automatisierungslösung oder die Möglichkeit, sich auf eine sich schnell ändernde Produktpalette einzustellen [MB00]. Diese Herausforderungen sind insbesondere unter ökonomischen Gesichtspunkten und in Anbetracht einer alternden Gesellschaft von großer Relevanz, da in einer alternden Gesellschaft weniger Arbeitskräfte zur Verfügung stehen.

Die Erstellung von Automatisierungslösungen, d. h. die Kombination von elektrischen, mechanischen und Software (Teil-)Systemen, unterliegt einem immer stärkeren Kostendruck [VHDF+14]. Durch die fortschreitende Integration von zusätzlichen Gewerken und Komponenten in die Automatisierungstechnik wird diese komplexer und damit schwerer zu beherrschen [MB00]. Dies führt zu einem steigenden Anteil der Engineeringkosten im Maschinen- und Anlagenbau. Zusätzliche Faktoren sind hohe Personalkosten in Deutschland oder der Bau von Unikaten, die immer wieder eine neue Planung und Entwicklung verlangen.

Ähnliche Problemstellungen
Wie beschrieben treten in der Automatisierungstechnik in verschiedenen Zusammenhängen wiederkehrende Problemstellungen auf, die sich mit den gleichen Prinzipien lösen lassen. Dies führt dazu, dass Implementierungen entstehen, die sich sehr ähneln, jedoch in wesentlichen Punkten voneinander abweichen. Im Rahmen der Arbeit des Arbeitskreises 2.2 der NAMUR[1] wurde festgestellt, dass es gängige Praxis ist, diese ähnlichen Lösungen nebeneinander zu entwickeln. Bestenfalls sind sich die Entwickler der Existenz der benachbarten Lösungen bewusst. Nicht ungewöhnlich ist allerdings auch, dass entstandene Implementierungen nur einem begrenzten Personenkreis bekannt sind. Folgen dieser Praxis sind:

- Duplizierung von Code,
- schlechte Wartbarkeit,
- hoher Aufwand bei der Lösungsentwicklung und -implementierung.

[1] Interessengemeinschaft Automatisierungstechnik der Prozessindustrie

Im Sinne der modernen Softwareentwicklung sind diese Folgen nicht erwünscht und stellen eine behebbare Herausforderung dar.

Komponentenbasierte Architekturen sind in der Automatisierungstechnik verbreitet
Aufgrund der komponentenbasierten Architekturen in der Automatisierungstechnik passiert es häufig, dass Komponenten oder ganze Subsysteme weiterentwickelt und damit verändert werden. Dies kann im ungünstigsten Fall dazu führen, dass die Gesamtlösung mit der neuen Version von verwendeten Komponenten nicht mehr funktionsfähig ist. Eine Konsequenz daraus ist, dass viel Zeit und Aufwand in die Entwicklung von sogenannten Migrationspfaden und Migrationstools investiert wird. Diese haben die Aufgabe, Automatisierungslösungen mit neuen Versionen von Komponenten oder Teilsystemen funktionsfähig zu halten. Wenn selbst entwickelte Komponenten verwendet werden, fehlen diese Hilfsmittel oft und die Aktualisierung der Automatisierung gestaltet sich schwierig.

Organisatorische Hindernisse bei der Wiederverwendung
Bei der Einführung eines Wiederverwendungskonzepts muss der Mensch berücksichtigt werden [Mey88]. Die besten Mechanismen und Werkzeuge sind wertlos, wenn die Nutzer sie nicht verwenden. Durch die Heterogenität der Nutzer in der Automatisierungstechnik ist die Sicherstellung der Akzeptanz besonders schwierig und wichtig. Bestehende Ansätze aus anderen Domänen (z. B. der Softwaretechnik) können nicht einfach übernommen, sondern müssen angepasst werden.

Die Herausforderungen bei der Wiederverwendung von Implementierungen sind sowohl technischer als auch organisatorischer Natur. Durch die Nutzung geeigneter Tools in Verbindung mit der Vorgabe, bestehende Implementierungen zu verwenden, können die organisatorischen Defizite teilweise durch technische Ansätze behoben werden. Mit dem in dieser Arbeit vorgestellten Konzept sollen die technischen und organisatorischen Herausforderungen gleichermaßen adressiert werden. Dies ist erforderlich, da die Fokussierung auf die Bearbeitung der technischen Herausforderung nicht zu einer Verbreitung von Implementierungen führt. Die potentiellen Nutzer müssen durch einen Mechanismus von bestehenden Implementierungen erfahren.

Diese aufkommenden Herausforderungen zu bewältigen, ist das Ziel vieler Initiativen und Arbeiten. Ein Ansatz dabei ist es, die Automation der Automation weiterzuentwickeln. Das bedeutet, dass nicht nur der (Produktions-)Prozess selbst automatisiert wird, sondern auch die Erstellung der Automatisierungslösung teil- oder vollautomatisiert abläuft. Modellbasierte Ansätze, mit denen z. B. die Struktur von Anlagen, Designpatterns für Lösungen oder Eigenschaften von Produkten beschrieben werden, sind ein möglicher Weg [FVHF+15, WKS+16]. Unter Einbeziehung und Kombination dieser Modelle wird die spezifische Automatisierungslösung generiert.

Allerdings ist auch die Verbesserung der Wiederverwendung und der Verbreitung bestehender Lösungen sinnvoll. Nur leicht unterschiedliche und sehr strukturiert aufgebaute Lösungen stellen eine gute Ausgangsbasis für die Wiederverwendung in der Automatisierungstechnik dar. Die Nutzung eines Variantenmanagements ist eine Möglichkeit für die Umsetzung der Wiederverwendung [VHDF+14]. Ebenso ist die Bekanntmachung und Verbreitung von bestehenden Lösungen eine Grundvoraussetzung für die Wiederverwendung, die typischerweise in der Automatisierungstechnik nicht gegeben ist und hergestellt werden muss.

1.2 Problemdefinition und Lösungsweg

Ziel der vorliegenden Arbeit ist die Entwicklung eines Konzepts zur Unterstützung der Wiederverwendung in komponentenbasierten Systemen. Die Komponenten selbst stellen durch ihren Einsatz in unterschiedlichen Systemen ein Mittel zur Wiederverwendung dar. Neben der Entwicklung von Komponenten wird ein hoher Aufwand in die Lösung von Automatisierungsaufgaben durch die Verknüpfung von Komponenten investiert. Der Fokus dieser Arbeit liegt daher auf der Wiederverwendung von Strukturen in komponentenbasierten Systemen. Da die Arbeiten an der Entwicklung von Automatisierungslösungen in der Regel nicht an einem Ort und nicht von einer Person durchgeführt werden, berücksichtigt das Konzept die dezentrale Wiederverwendung. Darunter ist die örtliche und/oder organisatorisch getrennte Entwicklung und Wiederverwendung von Automatisierungslösungen zu verstehen.

Damit die Wiederverwendung von Lösungen gelingt, müssen die folgenden drei Fragen, die an die Ausführungen in [Mey09] angelehnt sind, beantwortet werden:

- Wie sieht ein Mechanismus für komponentenbasierte (Teil-)Lösungen aus, so dass sie wiederverwendet und auf andere Anwendungsfälle übertragen werden können?
- Wie wird der Wiederverwendungsmechanismus in bestehende Tools und Prozesse integriert, so dass der Nutzer ihn verwendet?
- Wie erfährt der Nutzer, dass eine geeignete Lösung existiert?

Diese drei Fragen werden in der vorliegenden Arbeit für die komponentenbasierten Architekturen der Automatisierungstechnik beantwortet. Es wird dafür ein Konzept zur Wiederverwendung vorgestellt. Das Konzept besteht aus einem Mechanismus zur Wiederverwendung und einer Architektur für dessen Anwendung. Im Folgenden wird das Konzept anhand der drei Fragen vorgestellt und ein Überblick über den gewählten Weg zur Unterstützung der Wiederverwendung gegeben.

Die Beschreibung der Variabilität von Produkten ist in der Softwaretechnik und dem Automobilbau eine bewährte Methode, die Wiederverwendung zu unterstützen. Die Variabilität kann auf verschiedene Arten beschrieben werden. Diese Arbeit stützt sich auf die Delta-Modellierung als theoretische Basis für die Beschreibung von Variabilität. Die Delta-Modellierung ist für die Anwendung in der Automatisierungstechnik besonders geeignet, da sie gut in bestehende Systeme integriert werden kann. Delta-Modelle beschreiben durch Operationen die Transformation eines Produkts in ein anderes. Als Operationen können das Hinzufügen, Entfernen und Konfigurieren der Komponenten des Produkts verwendet werden. Zusätzlich können Verbindungen zwischen den Komponenten angelegt und gelöscht werden. Die Delta-Modelle stammen aus der Softwaretechnik und werden in diesem Bereich direkt auf Komponenten angewendet. Für die Nutzung in den hybriden Systemen der Automatisierungstechnik wird in dieser Arbeit ein Modell für die Beschreibung von Komponenten verwendet. Dieses stellt die Basis dar, auf der die Delta-Modelle angewendet werden. Durch die Beschreibung der Produkte (Lösungen von Automatisierungsaufgaben) durch Delta-Modelle existiert ein Mechanismus für die Wiederverwendung von Strukturen. Die durch Delta-Modelle dargestellten Strukturen können auf andere Anwendungsfälle angewendet werden und tragen so zur Wiederverwendung bei. Die Verwendung von Modellen

für die Lösung von Problemen ist in der Automatisierungstechnik ein erprobter Ansatz, der auch in dieser Arbeit Verwendung findet. Die Komponenten- und Delta-Modelle werden dafür in Laufzeitumgebungen realisiert, die eine Erkundung und Manipulation der Modelle ermöglichen.

Die dritte Frage beschäftigt sich mit der Information der Nutzer über vorhandene Lösungen. Diese Lösungen sind die Komponentensysteme, die eine Automatisierungsaufgabe lösen. Das beste Verfahren zur Wiederverwendung ist unwirksam, wenn existierende Lösungen nicht bekannt sind. In der vorliegenden Arbeit wird dafür ein zentraler Server für die Verwaltung der Komponentensysteme vorgeschlagen. Dieser Server verkörpert die „Gelben Seiten“ für die bestehenden Lösungen. Nutzer können dort nach Lösungen für ihre Probleme suchen und diese wiederverwenden. Kann keine passende Lösung gefunden werden, besteht die Möglichkeit, eine bestehende Lösung für den konkreten Anwendungsfall anzupassen. Ein Anwendungsfall ist der konkrete Kontext in dem eine Lösung verwendet oder wiederverwendet wird. Diese neue Lösung wird auf den Server geladen und steht für die erneute Nutzung bereit. Auf dem Server werden die Lösungen durch Delta-Modelle und Komponenten-Modelle in einer implementierungsunabhängigen Form abgelegt. Jedes Delta-Modell repräsentiert ein Produkt bzw. eine Lösung. Die Delta-Modelle werden dafür als Baum verknüpft. Die initialen Produkte bilden die Wurzeln des Baums. Die von diesen abgeleiteten Produkte werden in Form von Delta-Modellen gespeichert. Für die Erzeugung eines Produkts werden die Operationen aller Delta-Modelle von einer Wurzel aus angewendet. Dafür werden die Komponenten-Modelle genutzt. Komponenten-Modelle bestehen aus einem Teil, der die Komponenten-Typen beschreibt und einem anderen Teil, der die daraus verbundenen Instanzen beschreibt. Diese beinhalten die vorhandenen Komponenten-Typen und ermöglichen durch die Anwendung der Delta-Modelle, die dazugehörigen Instanz-Modelle zu erzeugen.

Zur Integration der beschriebenen Mechanismen in die bestehenden Systeme (Frage 2), wird ein Client vorgeschlagen. Dieser ermöglicht es, die Produkte (Lösungen der Automatisierungsaufgaben) vom Server herunterzuladen und in den lokalen Kontext einzubinden. Dafür wird auf dem Server das Instanz-Modell aus den Delta-Modellen erzeugt und auf den Client übertragen. Für die Übertragung werden die Instanz-Modelle serialisiert und de-serialisiert. Auf dem Client existiert ein lokales Modell, das beschreibt, wie die Komponenten des Modells realisiert werden. Durch Nutzung dieses Zusammenhangs wird das lokale Komponentensystem erzeugt.

Das vorgestellte Konzept wurde für das Laufzeitsystem des Lehrstuhls für Prozessleittechnik ACPLT/RTE realisiert. Zur Demonstration der Funktionsfähigkeit des Konzepts wird es auf PID-Regler, auf die Prozessführungskomponenten des Lehrstuhls und für die modulare Anlage des Lehrstuhls angewendet.

1.3 Aufbau der Arbeit

Die Gliederung der vorliegenden Arbeit ist in Abbildung 1.2 dargestellt. In Kapitel 2 werden die Grundlagen der Automatisierungstechnik und der Stand der Technik im Hinblick auf komponentenbasierte Architekturen erläutert. In diesem Rahmen wird die in dieser

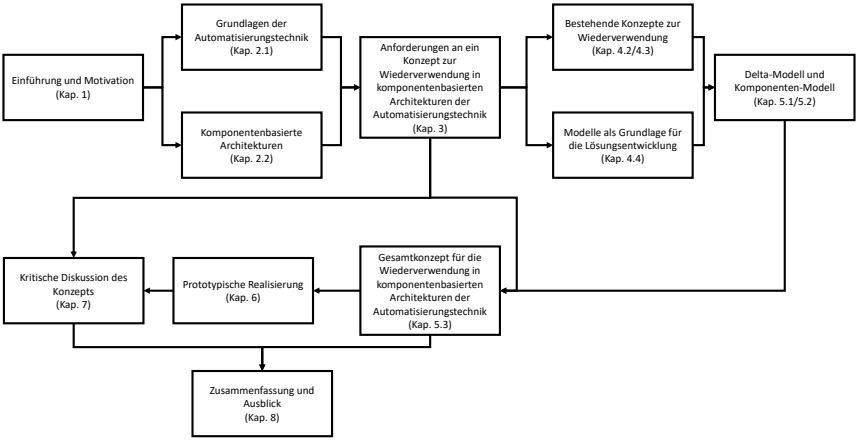


Abbildung 1.2: Darstellung des Aufbaus der vorliegenden Arbeit.

Arbeit verwendete Definition einer Komponente vorgestellt und mit anderen Definitionen aus der Domäne und der Softwaretechnik in Verbindung gesetzt. Es wird gezeigt, dass komponentenbasierte Architekturen in der Automatisierungstechnik weit verbreitet sind.

Die Anforderungen an das Konzept zu Wiederverwendung werden in Kapitel 3 vorgestellt. Ausgangspunkt der Betrachtung sind die Besonderheiten der Automatisierungstechnik. Im Anschluss daran werden die funktionalen und nicht-funktionalen Anforderungen an das Konzept eingeführt.

Kapitel 4 bietet einen Überblick über den aktuellen Stand der Wissenschaft. Ausgehend von einer Vorstellung der eigenen Vorarbeiten werden zunächst die Grundlagen der Wiederverwendung vorgestellt. Der Schwerpunkt liegt auf den Arten der Wiederverwendung und der Betrachtung der Versionierung. Dem schließt sich ein Überblick über die Grundlagen der Variantenbeschreibung an. Anschließend werden die Delta-Modelle aus der Softwaretechnik vorgestellt. Am Ende des Kapitels werden die Grundlagen der Modellierung und Beispiele aus der Automatisierungstechnik dargestellt.

Das Konzept für die Wiederverwendung in komponentenbasierten Architekturen wird in Kapitel 5 eingeführt. Am Anfang des Kapitels werden das Komponenten- und das Delta-Modell vorgestellt. Danach wird deren Verwendung in einem Gesamtkonzept dargestellt. Dabei wird auf die dezentrale Wiederverwendung und die dafür nötigen Prozesse eingegangen. Abschließend werden die Randbedingungen und die Vorteile des Konzepts betrachtet.

In Kapitel 6 wird die prototypische Realisierung vorgestellt und anhand von drei Anwendungsfällen der Nutzen des Konzepts verdeutlicht. Kern der in ACPLT/RTE umgesetzten Implementierung sind die vorgestellten Metamodelle und die Transformationen zur automatisieren Erzeugung der Modelle.

Ein Überblick und die Diskussion der Ergebnisse der Arbeit ist in Kapitel 7 zu finden. Abschließend wird in Kapitel 8 ein Ausblick auf mögliche weiterführende Arbeiten gegeben.

2 Grundlagen und Stand der Technik

In diesem Kapitel wird ein Überblick über die relevanten Arbeiten und über den Stand der Technik gegeben. Zunächst werden die Grundlagen der Automatisierungstechnik erläutert. Dabei wird auf die dezentralen Leitsysteme und modulare Anlagen in der Prozessindustrie näher eingegangen, bevor die Prozesse zum Bau einer Anlage näher erläutert werden.

2.1 Grundlagen der Automatisierungstechnik

> Es ist das Ziel der Automatisierungstechnik, Systeme so zu steuern, dass sie autonom arbeiten. [Lun03]

Aus diesem Zitat von Lunze geht hervor, dass sich die Automatisierungstechnik mit dem Steuern von Systemen beschäftigt. Die Art der Systeme wird dabei nicht eingeschränkt. Die Automatisierungstechnik kann anhand des betrachteten Systems klassifiziert werden. Eine gängige Klassifikation ist die Unterteilung in Produktautomatisierung und Produktionsautomatisierung. Die erst genannte beschäftigt sich mit Produkten (z. B. Autos, Flugzeugen, Hausgeräte). Gegenstand der Produktionsautomatisierung ist die Automation der Herstellungsprozesse von Produkten. In der nächst feineren Klassifikation wird die Produktionsautomatisierung in Fertigungsautomation (Fertigung von Stückgütern) und Prozessautomatisierung (Batch- und Konti- Prozesse, z. B. mit Gasen, Schüttgut, Flüssigkeiten oder Aluminiumbändern) unterteilt. Jede der beiden Gruppen hat spezifische Eigenschaften und Anforderungen an die Automatisierungssysteme [FA09]. Gegenstand der vorliegenden Arbeit ist die Produktionsautomatisierung mit dem Schwerpunkt auf der Prozessautomatisierung.

Nach Lunze ist das Ziel der Steuerung von Systemen, die Autonomie des Systems zu erreichen. Der Grad der Autonomie hängt vom jeweiligen Anwendungsszenario ab und kann ebenso zur Klassifikation genutzt werden [TE18, Gas12]. Die Bandbreite reicht von einem System ohne Automation bis hin zu einem System, das ohne menschliche Unterstützung funktioniert. Allerdings ist der Mensch weiterhin ein wichtiger Bestandteil der Automatisierung [VHDB13, Lun03]. Steigt der Grad der Autonomie, wird zwar die Selbstständigkeit der Systeme erhöht, jedoch ist der Nutzer weiterhin in überwachender Funktion erforderlich. Zusätzlich wird der Mensch für die Entwicklung und den Aufbau von Automatisierungslösungen benötigt. Dies gilt gleichermaßen für die Wartung und Instandhaltung der verbauten Systeme.

In Abbildung 2.1 ist der Aufbau eines automatisierten Systems dargestellt. Das System besteht aus dem Nutzer, der Nutzerschnittstelle (Mensch-Prozess-Kommunikation), dem Automatisierungssystem und dem technischen Prozess. Bestandteil des Automatisierungs-

systems ist mindestens eine Hardwarekomponente, auf der das Steuerungsprogramm ausgeführt wird. Die Hardware verfügt über Ein- und Ausgänge zum technischen Prozess. Die Sensoren und Aktoren werden als Schnittstelle zum technischen Prozess [Lun03] bzw. zwischen physischer Welt und Informationswelt [KE12] verstanden. In [HSF+13] wird herausgestellt, dass das Besondere an der Automatisierungstechnik die Domänen-übergreifende Betrachtung des Systems ist. So müssen alle Bestandteile perfekt zusammen funktionieren, damit das Gesamtsystem wie geplant arbeitet [VHDF+14]. Jede Domäne verfügt über eine andere Sicht auf den Betrachtungsgegenstand. Die Verfahrenstechnik hat die Aufgabe Anlagen zu planen und kann so den Aufbau der Anlage beeinflussen [HSF+13]. Für die Automatisierungstechnik ist beispielsweise der Anlagenaufbau eine Randbedingung, die berücksichtigt werden muss. Dadurch steigt die Komplexität dieser Systeme zunehmend an [VHDF+14].

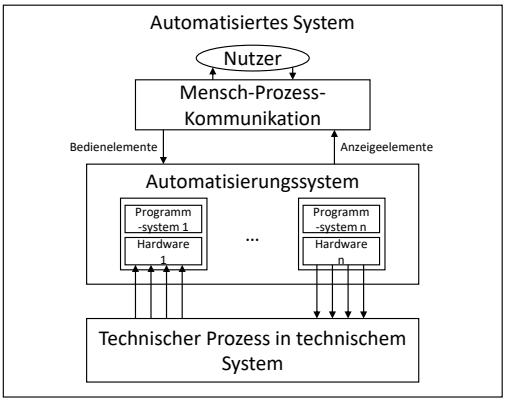


Abbildung 2.1: Aufbau eines automatisierten Systems nach [VHDFG13, VHDB13]

Im Folgenden wird der Aufbau von Automatisierungssystemen in der Prozessindustrie, d. h. von (Prozess-) Leitsystemen, näher beschrieben. Anwendungszweck eines Leitsystems ist die Führung eines Prozesses entsprechend gegebener Randbedingungen (d. h. die Umsetzung der Prozessführung). Unter Prozessführung sind nicht nur die Regelung und Optimierung einer Anlage zu verstehen, sondern darüber hinaus alle zielgerichteten Maßnahmen, um den jeweiligen Produktionsprozess zu beherrschen und entsprechend ihrer Zielvorgaben zu fahren. Diese Definition schließt ausdrücklich die Tätigkeit des Anlagenfahrers ein und erklärt die Prozessführung zu einer Aufgabe, an der unterschiedliche technische Disziplinen, u. a. die Verfahrenstechnik und die Automatisierungstechnik (vgl. [KBD+08, PE94]), beteiligt sind. Im Folgenden werden Systeme zur Prozessführung und Automatisierungslösung synonym verwendet.

Zunächst werden die in der industriellen Anwendung verbreiteten dezentralen Prozessleitsysteme beschrieben. Anschließend wird auf die aufkommenden modularen Anlagen und damit die Package Units in der Prozessindustrie als Beispiel für die Modularisierung der Produktion eingegangen.

2.1.1 Aufbau von Dezentralen Prozessleitsystemen

Dezentrale Prozessleitsysteme bilden die Grundlage der modernen Industrieautomation. Nachfolgend wird erst der Aufbau der Hardwarekomponenten beschrieben, anschließend wird ein Überblick über die Softwareseite der Automatisierungssysteme gegeben. Der Fokus liegt dabei auf der Architektur der Software und der Sprachen aus der IEC 61131 [IEC14b]. Zusätzlich wird ein Blick auf die IEC 61499 [IEC05] geworfen und deren Softwarearchitektur vorgestellt.

Aufbau der Hardwarekomponenten

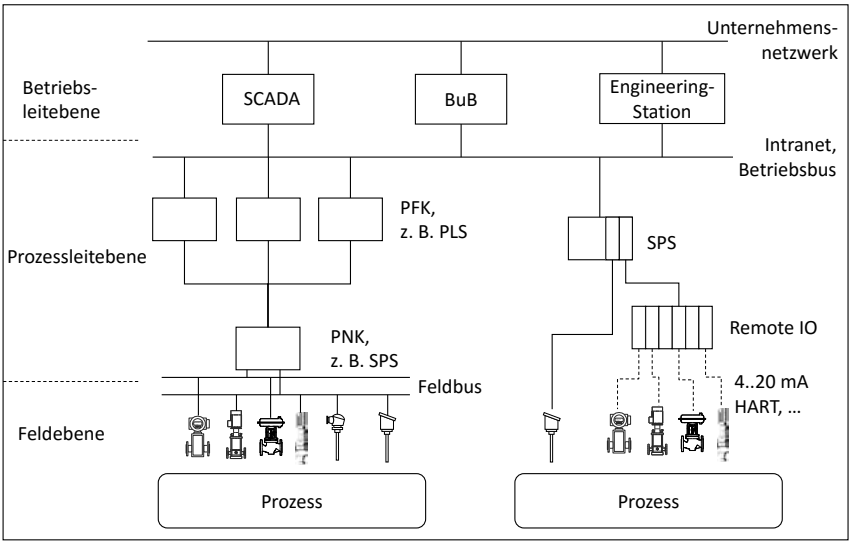


Abbildung 2.2: Architektur eines Prozessleitsystems nach [KCJ+10]

In Abbildung 2.2 ist schematisch der Aufbau eines dezentralen Prozessleitsystems dargestellt. Die Abbildung ist analog zur Automatisierungspyramide aufgebaut und beginnt am unteren Ende mit dem zu automatisierenden Prozess. Direkten Kontakt mit dem Prozess haben die Sensoren und Aktoren. Die Sensoren haben die Aufgabe, Informationen über den aktuellen Zustand des Prozesses zu sammeln. Beispiele für Sensoren sind Füllstands- oder Durchflussmessgeräte. Die Aktoren werden verwendet, um den Prozess zu beeinflussen. Ventile oder Pumpen sind beispielsweise Aktoren.

Die von Sensoren gemessenen Werte und die Vorgaben, wie sich die Aktoren zu verhalten haben, werden über Kommunikationssysteme an bzw. von prozessnahen Komponenten übermittelt. Verwendung findet die Zweidrahtanbindung, bei der die Feldgeräte direkt mit

den prozessnahen Komponenten verbunden sind und Messwerte und Stellwerte als analoges Stromsignal (4 bis 20 mA) übertragen werden. Das HART Protokoll ermöglicht es, zusätzlich zu den analogen Werten Konfigurationsdaten der Feldgeräte zu übermitteln und so Informationen über die Feldgeräte zu erhalten oder sie zu parametrieren. Eine weitere Form der Anbindung sind Feldbusse. Hierbei werden die Feldgeräte mit prozessnahen Komponenten über einen digitalen Bus verbunden. Die Feldbusanbindung ermöglicht es, die Feldgeräte mit erweiterten Funktionen auszustatten und so zu intelligenten Geräten aufzuwerten.

Prozessnahe Komponenten sind heutzutage fast ausnahmslos Speicherprogrammierbare Steuerungen (SPS) oder Remote-IOs. Die SPS verfügen über Schnittstellen für die Busse oder die Zweidrahtanbindung. Die Feldgeräte sind entweder direkt an die SPS oder die Remote-IO angeschlossen. Eine Remote-IO hat die Aufgabe, die Signale von und zu Feldgeräten über die Zweidrahtanbindung aufzunehmen und in ein Bussignal umzuwandeln. Über diesen Feldbus ist die Remote-IO mit einer SPS verbunden, die die Weiterverarbeitung der Werte übernimmt. Verglichen mit Feldbussen stellen Systembusse höhere Anforderungen an die verfügbare Bandbreite. Im Gegensatz dazu sind die Echtzeitanforderungen nicht so hoch [FA09, ASE08].

Steuerungs- und Regelungsaufgaben werden von prozessnahen Komponenten realisiert. Aufgaben, die mit dem Koordinieren und Optimieren des Prozesses assoziiert sind, werden dagegen von prozessfernen Komponenten (z. B. Prozessleitsystem (PLS)) ausgeführt. Das können höhere Regelungskonzepte, wie Model Predictive Control (MPC) oder andere Optimierungsverfahren sein. Zu den prozessfernen Komponenten gehören auch Systeme, die für die Aufzeichnung und Archivierung von Mess- und Stellwerten verantwortlich sind.

Die Komponenten oberhalb der Feldgeräte bis hin zu den prozessfernen Komponenten bilden die Prozessleitebene in der Automatisierungspyramide. Oberhalb der prozessfernen Komponenten beginnt mit der Verwendung des Betriebsbusses bzw. des Intranets die Betriebsleitebene. Auf dieser Ebene sind die Engineeringstationen für das Konfigurieren und Warten des Leitsystems und die Stationen für das Bedienen und Beobachten (BuB) des Prozesses angesiedelt. Daneben stehen auch leistungsfähige Manufacturing Execution Systems (MES) und Enterprise Resource Planning (ERP) Systeme zur Verfügung. Das Intranet bzw. der Betriebsbus sind über Firewalls an Büronetze und an das Internet angeschlossen [KCJ+10]. Dies ermöglicht einen kontrollierten lesenden Zugriff, damit beispielsweise eine Fernwartung oder Analysen und Optimierungen von ortsfernen Experten durchgeführt werden können. Zusätzlich können so, über die Grenzen von Standorten hinweg, Leistungsindikatoren sichtbar gemacht werden und Prozesse und Produktionsanweisungen (Rezepte) im Unternehmen verbreitet werden.

Speicherprogrammierbare Steuerungen und Laufzeitsysteme

Die SPS entwickelten sich aus den verbindungsprogrammierten Steuerungen, bei denen die Funktionalität mittels Schütz- und Relaistechnik realisiert wird [FA09]. Aktuell sind SPS sowohl in der Prozess- als auch in der Fertigungsautomation sehr verbreitet und es gibt sie in verschiedenen Leistungsklassen. Durch die zunehmende Verteilung von Funktionalität direkt ins Feld und die zunehmende Verwendung von PCs (Soft-SPS) für das Steuern und

Regeln geht die Bedeutung von großen SPS zurück. Kleinere SPS kommen als Steuerungen von einzelnen Prozessmodulen zum Einsatz.

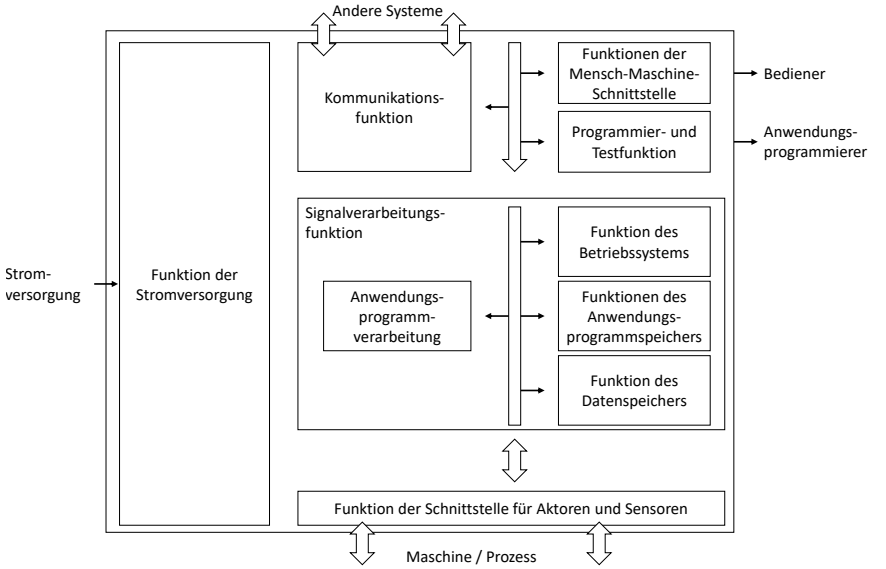


Abbildung 2.3: Darstellung des funktionalen Aufbaus einer SPS nach [IEC04]

Abbildung 2.3 zeigt den funktionalen Aufbau einer SPS. Es sind die vier großen Funktionsbereiche zu erkennen: Stromversorgung, Schnittstelle zu Sensoren und Aktoren (Feld), Kommunikation und Signalverarbeitung. Wie beschrieben, gibt es unterschiedliche Möglichkeiten, eine SPS an die Sensoren anzubinden. Für die interne Verwendung der Prozessinformationen bzw. der Beeinflussung der Aktoren abstrahiert die Schnittstelle zu den Sensoren und Aktoren von den unterschiedlichen Anbindungsarten und stellt ein einheitliches Interface zur Verfügung. Die Signalverarbeitungsfunktion ist das eigentliche Herzstück der SPS. Sie besteht aus der Anwendungsprogrammverarbeitung, die unter Einbeziehung des Betriebssystems und des Datenspeichers die Anwendungsprogramme ausführt. Dabei findet eine zyklische Auswertung der Eingänge zum Feld statt und es werden durch die Anwendungsprogramme die Ausgänge gesetzt. Die Kommunikation zu anderen Systemen, sowie zum Bediener und Anwendungsprogrammierer, erfolgt über die Kommunikationsfunktion.

Nach [NAM02] setzt sich das Betriebssystem von Leitsystemen (SPS) aus einem Standardbetriebssystem und einem Leitsystem-Betriebssystem zusammen. Ein Betriebssystem- oder Leitsystemhersteller stellt das Standardbetriebssystem bereit. Dieses stellt die Basis der Software dar. Auf ihm arbeitet das Leitsystem-Betriebssystem des Leitsystemherstellers, in dem die Anwendungen für die Automatisierung ausgeführt werden. Das Leitsystem-Betriebssystem kann auch als Laufzeitumgebung (runtime system oder runtime environment) bezeichnet werden. Dieser Begriff stammt aus der Informatik und beschreibt ein

System, das eine Schicht zwischen dem Betriebssystem und der Applikation des Anwenders bildet [App90, Grü17]. Das Laufzeitsystem stellt Funktionen für den Zugriff auf die Hardware, für das Speicherhandling (Allokation und Freigabe), für die Interaktion mit dem System (z. B. Debuggen), für die Introspektion und für die Reflexion bereit (vgl. [Grü17]). Laufzeitsysteme sind nicht zwingend mit PLS und SPS assoziiert, sondern finden auch, sofern es die Randbedingungen zulassen, in Soft-SPS oder als Applikation auf einem normalen PC Verwendung.

Softwarearchitekturen

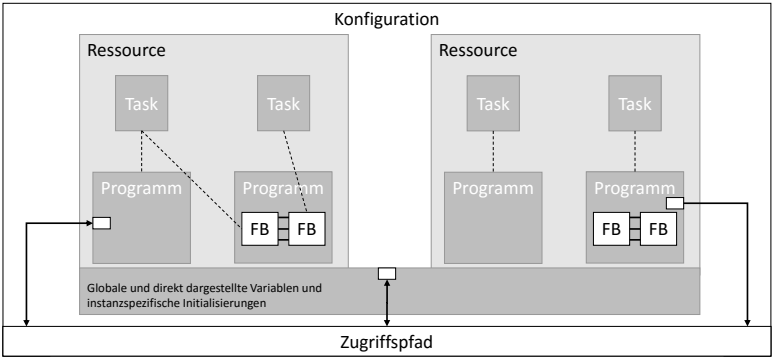


Abbildung 2.4: Darstellung des Softwarearchitektur einer SPS nach [IEC14b].

In der Automatisierung ist die in Abbildung 2.4 dargestellte Softwarearchitektur aus der IEC 61131 weit verbreitet. Die Architektur stellt die Softwaresicht auf eine Norm-konforme Umgebung für die Realisierung von Automatisierungsfunktionen dar. Die Konfiguration ist mit einer SPS assoziiert. Innerhalb der Konfiguration existieren Ressourcen, die die beschriebenen Funktionen des Laufzeitsystems bereitstellen. Die Ressource ist die Umgebung für die Programme und Tasks. Programme enthalten die ausführbare Logik (z. B. als Netz von Funktionsbausteinen) und durch sie erfolgt der Zugriff auf den Speicher bzw. die Peripherie. Einer oder mehrere Tasks koordinieren die Ausführung der Programme bzw. der einzelnen Funktionsbausteine. Die Ausführung erfolgt zyklisch, wobei die Zykluszeit durch die Wahl der Tasks vom Anwender eingestellt werden kann. Nach [JT00] existiert in der Norm zusätzlich der Begriff der Programmorganisationseinheit (POE). Darunter wird die kleinste unabhängige Softwareanwendung in dieser Architektur verstanden [Grü17].

Der Austausch von Informationen innerhalb von Programmen erfolgt durch die direkte Verknüpfung von beispielsweise Funktionsbausteinen. Für die Kommunikation zwischen Programmen derselben Ressource und zwischen verschiedenen Ressourcen stehen globale Variablen zur Verfügung, die von allen Programmen in einer Konfiguration gelesen und geschrieben werden können.

Als Erweiterung der vorgestellten Softwarearchitektur ist die IEC 61499 [IEC05] zu sehen. Sie definiert eine Architektur für verteilte Systeme, bei der eine Anwendung auf mehr als

einer SPS ausgeführt werden kann. Im Gegensatz zu den beschriebenen Tasks, die eine zyklische Abarbeitung der Programmlogik vorsehen, werden Funktionsbausteine nach der IEC 61499 Ereignis-gesteuert ausgeführt. Zusätzlich dazu definiert die Norm, additiv zu den Verbindungen für den Informationsfluss, Verbindungen für Ereignisse (Events) zwischen Funktionsbausteinen. Die dritte Neuerung ist die Definition von Diensten für die Interaktion mit und zwischen Ressourcen. Hierzu werden Konfigurations- und Kommunikationsdienste sowie Interfaces für die entsprechenden Bausteine festgelegt.

Sprachen der IEC 61131

Für die Programmierung der benötigten Funktionalitäten stehen in einer IEC 61131 konformen Umgebung fünf Programmiersprachen zur Verfügung. Anweisungsliste (AWL) und Strukturierter Text (ST) sind zwei textuelle Programmiersprachen. Im Gegensatz dazu sind die Funktionsbausteinsprache (FBS), der Kontaktplan (KOP) und die Ablaufsprache (AS) grafische Programmiersprachen. Die folgende Vorstellung der Programmiersprachen ist [JT00, Grü17, Kam17] entnommen.

Anweisungsliste: AWL wird als maschinennahe Programmiersprache eingestuft. Eine Anweisung besteht aus einem Operator und einem Operanden. Für die Festlegung des Programmablaufes werden Sprungmarken am Anfang der Zeilen verwendet. AWL dient als gemeinsame „Zwischensprache“ für sowohl die textuellen als auch die grafischen Sprachen der IEC 61131.

Strukturierter Text: ST wird im Kontext der IEC 61131 als Hochsprache bezeichnet und besteht aus Anweisungen zum Programmablauf (z. B. FOR- und WHILE-Schleifen), sowie Operatoren/Funktionen und Operanden. Im Vergleich zu AWL hat ST die Vorteile, dass eine sehr kompakte Formulierung und einen übersichtlichen Aufbau des Programms erlaubt wird. Nachteilig ist, dass die Übersetzung in Maschinencode nicht direkt beeinflussbar ist und dass es durch die höhere Abstraktionsstufe zu einem Verlust an Effizienz kommen kann.

Funktionsbausteinsprache: FBS stammt aus der Einzelgerätetechnik bzw. der Signalverarbeitung und ist im Gegensatz zu AWL und ST eine grafische Programmiersprache. Grafische Elemente der Sprache sind die Bausteine, Verbindungen, Konnektoren und Elemente für die Ausführungssteuerung. Die Bausteine werden in Funktionen (ohne internen Speicher) und Funktionsbausteine (mit internem Speicher) unterteilt. Offene Eingänge von Bausteinen können mit Variablen oder Konstanten beschaltet werden. Die Elemente der Sprache werden zu Funktionsbausteinnetzen zusammengefasst.

Kontaktplan: Analog zu der Funktionsbausteinsprache kommt KOP historisch aus dem Bereich der elektromechanischen Relaissysteme. Daher wird der „Stromfluss“ durch die Netzwerke beschrieben und es werden im Wesentlichen boolesche Signale verarbeitet. Die Basis der KOP-Netzwerke sind zwei „Stromschienen“, die auf der linken und rechten Seite die Pläne begrenzen. Auf der linken Schiene herrscht der logische Zustand 1 und durch Verbindungen zwischen den Schienen kann der Strom abhängig vom Zustand von Variablen fließen. So können logische Bedingungen aus Parallel- und Reihenschaltungen aufgebaut werden.

Ablaufsprache: Die AS dient dazu, eine komplexe Aufgabe in überschaubare Unteraufgaben zu zerlegen. Ein Beispiel für ein derartiges Vorgehen sind chemische Prozesse, die aus Zwischenschritten aufgebaut sind. In der Prozessindustrie könnten das beispielsweise die folgenden Schritte sein: Befüllen, Heizen, Rühren, Ablassen. Dafür werden durch die Sprache Schritte und Transitionen bereitgestellt. In den Schritten können Aktionen (z. B. das Setzen von Variablen) ausgeführt werden. In den Transitionen werden Bedingungen für den Wechsel von einem Schritt in den anderen geprüft. Diese Bedingungen können das Ablaufen eines Timers oder der Zustand einer Variablen, also der Zustand des Prozesses, sein.

Prozessführungsarchitekturen

Das übergeordnete Ziel der Automatisierungstechnik ist die Steuerung und Überwachung von Prozessen durch die Automatisierungslösung. Die Grundlage dafür bilden die beschriebene Hard- und Software. Für den Gesamterfolg muss unter deren Verwendung eine sogenannte Prozessführung entwickelt werden. Die wesentliche Aufgabe der Prozessführung ist die Bereitstellung der Steuerungslogik für die jeweilige Aufgabe. Im Folgenden wird der schematische Aufbau dieser Steuerungslogik aus einer funktionalen Sicht vorgestellt.

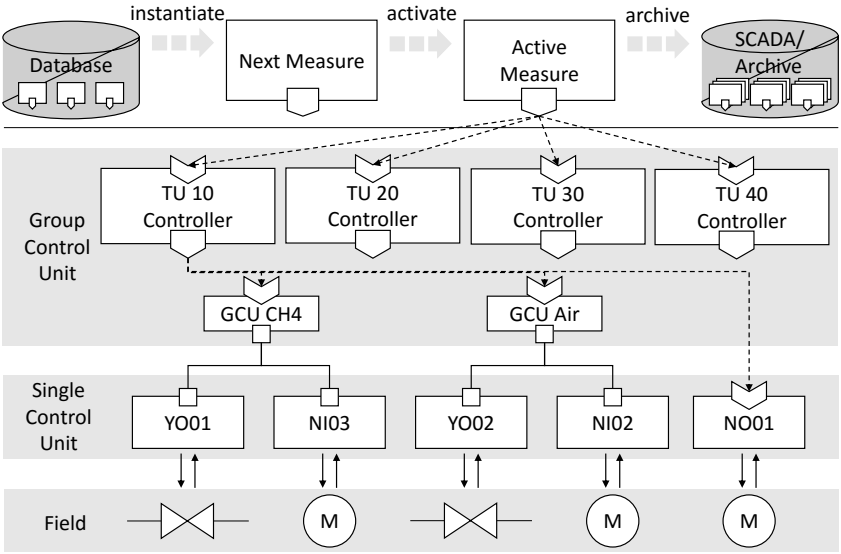


Abbildung 2.5: Darstellung einer hierarchischen Prozessführung [WTPE17].

In [PE94] wird ausgehend von einem hierarchischen Führungsmodell eine Prozessführungsarchitektur vorgestellt. Deren fundamentale Bestandteile sind die Prozessführungseinheiten. Diese kapseln die jeweiligen Funktionalitäten und bieten eine Schnittstelle an, um Pro-

zessführungsaufträge entgegenzunehmen bzw. zu versenden. Je nach Position innerhalb der Hierarchie werden diese Prozessführungseinheiten als *Einzelsteuereinheiten* oder *Gruppensteuereinheiten* bezeichnet. Einzelsteuereinheiten kapseln einzelne Aktoren im Leitsystem. Gruppensteuereinheiten aggregieren die Funktionalitäten und synchronisieren die Aktionen der ihnen zugeordneten Einzelsteuereinheiten.

In Abbildung 2.5 ist der Aufbau einer hierarchischen Prozessführung dargestellt. Am unteren Rand sind die Aktoren der Feldebene zu erkennen. Diese werden im Leitsystem durch sogenannte Einzelsteuereinheiten (Single Control Unit) repräsentiert (z. B. *YO01*). Neben der untersten Ebene der Steuerung, beispielsweise Verriegelungen[1], bilden die Einzelsteuereinheiten zusätzlich auch den Zugangspunkt für die Interaktion mit dem Aktor aus dem Leitsystem heraus. Alle Vorgaben, wie Sollwerte oder Parameter, werden über die Einzelsteuereinheiten eingestellt. Wenn Informationen über den Aktor benötigt werden, stellt sie die Einzelsteuereinheit bereit. Oberhalb der Einzelsteuereinheiten befinden sich die Gruppensteuereinheiten (Group Control Units). Ihre Aufgabe ist es, die von ihnen kontrollierten Einzelsteuereinheiten zu koordinieren. In diesem Beispiel (Abbildung 2.5) kontrolliert *GCU_CH4* den Motor *NI03* und das Ventil *YO01*. Durch die Aggregation der Funktionalitäten kann der Durchfluss eingestellt werden. Die Gruppensteuereinheit fungiert als virtueller Aktor und kapselt nach oben diese Funktionalität. Gruppensteuerungen können beliebig hoch übereinandergestapelt, d. h. deren Funktionalität aggregiert, werden. In Abbildung 2.5 wird eine Gruppensteuereinheit für die Teilanlage TU10 (*TU10 Controller*) umgesetzt. In der Abbildung sind zwei Arten der Verbindung zwischen Prozessführungskomponenten zu erkennen. Von *GCU_CH4* nach unten zu *YO01* wird eine feste Verdrahtung verwendet. Nach oben zu *TU10 Controller* werden Prozessführungsaufträge bzw. Prozessführungsdienste genutzt. Zur Realisierung von prozeduralen Produktionsprozessen, die ein Produkt unter Einbeziehung verschiedener Anlagenteile produzieren, sind sogenannte Maßnahmen vorgesehen. In der Prozessindustrie werden diese auch als Rezepte bezeichnet. Eine Maßnahme beinhaltet eine Prozedur und die Schnittstellen zur Interaktion mit den Prozessführungskomponenten. Maßnahmen können nach ihrer Durchführung zu Dokumentationszwecken zusammen mit den relevanten Prozessparametern gespeichert werden.

Die lose Kopplung durch die Verwendung von Prozessführungsdiensten schafft einen hohen Freiheitsgrad bei der Verteilung der Komponenten auf unterschiedliche Hardware. Die Verteilung von Einzelsteuereinheiten ist jedoch nur eingeschränkt möglich, da diese einen Zugang zum jeweiligen Aktor benötigen. In [WE15a] wird eine Realisierung dieser Schnittelle auf Basis von Nachrichten vorgestellt. Dabei werden die Aufträge in Klartext verschickt und ermöglichen so eine einfachere Nachverfolgung und Fehlerbehebung.

Die so aufgebaute Prozessführung kann in einen prozessnahen Teil (Maßnahmen) und einen anlagennahen Teil (Steuereinheiten) aufgeteilt werden. Die Steuereinheiten aggregieren die Funktionalität der Einheiten unter ihnen und die Maßnahmen zerlegen den Prozess in von der Anlage bearbeitbare Teilaufgaben. In der Phase des Entwurfs muss festgelegt werden, welche Aufgaben von welchem Teil realisiert werden. Denkbar ist, dass die Steuereinheiten die Funktionalität so weit aggregieren, dass die Anlage durch die oberste Steuereinheit komplett gesteuert werden kann. Alternativ können die Maßnahmen direkt auf die Geräte wirken. Theoretisch sind alle Abstufungen zwischen diesen Extremen denkbar, je-

[1] Verriegelung bedeutet das Festhalten oder Überführen eines Aktors in einen sicheren Zustand.

doch muss die höhere Flexibilität (Maßnahmen) gegen die einfachere Ansteuerung (höhere Aggregation der Einzelsteuereinheiten) abgewogen werden. In [WTPE17] wird mit dem Betriebsmittel-Maßnahmen Modell ein Ansatz vorgestellt, diese Entscheidung für jeden Anwendungsfall spezifisch zu treffen.

In [UDKO12] wird eine Architektur für die Integration von Modulen einer Anlage zu einer Gesamtanlage beschrieben. Ziel der Architektur ist eine einfache Zusammenfassung der Module durch die Nutzung einer Dienstschnittstelle. Diese erlaubt den Zugriff auf die Funktionalität und den Status der einzelnen Module. Dieser Ansatz wird in [BFK+17] für die Verwendung von Micro-Services ausgebaut. Micro-Services erlauben den Zugriff auf die Funktionalität durch eine Reihe einfacher Dienste, die vom Client in geeigneter Weise verwendet werden. Die Dienstaufrufe können so einfach gehalten werden, wohingegen Server und Client smart sein müssen.

2.1.2 Package Units und Modulare Anlagen in der Prozessindustrie

Als Beispiel für ein System in der Automatisierungstechnik, das mehrfach eingesetzt werden soll, werden im Folgenden die Package Units und modulare Anlagen vorgestellt. Diese bestehen aus Software- und Hardwareelementen. Da sie mehrfach eingesetzt werden, bieten die Module einen guten Anwendungsfall für Wiederverwendungskonzepte.

Der Arbeitskreis 1.6 „Package Unit“ der NAMUR[2] definiert *Package Units* als Anlagenteile, die sowohl alleine funktionsfähig als auch abgeschlossen sind und wiederholt die gleiche Funktionalität bereitstellen [NAM96]. Sie werden dafür von einem Hersteller vertrieben, können jedoch verschiedene Komponenten anderer Leistungserbringer beinhalten. Package Units sind vorkonfektioniert und bieten die Möglichkeit, einen abgegrenzten Funktionsumfang von einer Teilanlage zur Verfügung gestellt zu bekommen. Die Frage, ob die Automatisierung der Package Units in ein übergeordnetes Leitsystem integriert oder, ob die Teilanlage separat (vor Ort) gefahren wird, ist nach [NAM96] eine Designentscheidung, die für jeden Einzelfall erneut abgewogen werden muss. Vorteile der Wiederverwendung von Package Units sind:

- Günstige Herstellungskosten,
- Große Erfahrung mit den Teilanlagen und entsprechend hohe Garantien für den Betreiber,
- Fokussierung auf die Planung des Gesamtprozesses.

Durch die Wiederverwendung entsteht ein spezifisches Wissen und der Hersteller kann seinen Produktionsprozess verbessern. So kann er die Herstellungskosten senken und gleichzeitig die Qualität verbessern [Die02, Lim94]. Bei einer höheren Zahl von verwendeten Package Units können mehr Informationen über das Verhalten und die Leistung der verbauten Einheiten erhoben werden. Dies erlaubt dem Hersteller sein Produkt zu verbessern und, zusätzlich zur gesteigerten Qualität des Produktionsprozesses, das Produkt selbst

[2]Die NAMUR ist ein internationaler Verband der Anwender von Automatisierungstechnik der Prozessindustrie.

genauer an die Kundenwünsche und den Verwendungszweck anzupassen. Neben den Verbesserungen auf der Herstellerseite und den günstigeren Kosten für den Anlagenbetreiber entsteht auch ein Potential für Verbesserung darin, dass sich der Anlagenplaner auf die Planung des Gesamtprozesses konzentrieren kann. Eine Standardfunktionalität kann so einfach und günstig hinzugekauft werden, ohne dass sich der Planer über die Details der Umsetzung kümmern muss.

Nachteilig ist, dass die zur Verfügung stehenden Package Units nicht auf alle Sonderwünsche oder Spezialanforderungen des Anlagenbetreibers eingehen. Auftretende Abweichungen, die bei der Produktion berücksichtigt werden müssen, führen zu einem erhöhten Entwicklungsaufwand und einem Verlust der skizzierten Vorteile [NAM96].

Ein nächster Schritt, die Vorteile der Package Units in der Verfahrenstechnik besser nutzbar zu machen, ist die Verwendung von modularen Anlagen. Modulare Anlagen und die sich daraus ergebenden neuen Anforderungen an die Automatisierungstechnik sind in der NAMUR Empfehlung 148 [NAM13] beschrieben.

In modularen Anlagen werden verschiedene sogenannte Module zu Anlagen zusammengefasst. Die Anlage besteht aus dem *Backbone*, der die gesamte von den Modulen benötigte Infrastruktur umfasst und Möglichkeiten zum Andocken für die Module bereitstellt. Die Infrastruktur umfasst die Versorgung mit Energie, Informationen und Edukten sowie den Abtransport der hergestellten Produkte. In den „Backbone“ werden die für den konkreten Anwendungsfall benötigten Module in der erforderlichen Menge und Reihenfolge eingebracht. Verschiedene Arten von einzelnen Modulen sind in [NAM13] vorgesehen. Es werden drei Eigenschaften von Modulen unterschieden: autonom, integrierbar und modular. Diese Eigenschaften sind nicht disjunkt. Autonome Module können autark betrieben werden und benötigen selbst keine Interaktion mit benachbarten Modulen. Daher muss der Backbone solch ein Modul von der Umgebung entkoppeln (z. B. durch die Bereitstellung von Pufferbehältern). Das Modul verfügt über eine eigene Automatisierung mit definierten Schnittstellen zur Erfassung von Betriebsdaten und für die Inbetriebnahme der modularen Anlage. Integrierbare Module sind darüber hinaus in ein übergeordnetes Leitsystem integrierbar, d. h., es existiert nicht nur eine Schnittstelle für die Interaktion, sondern das Leitsystem kann eine modulübergreifende Steuerung realisieren. Die Integration muss auf verschiedenen Ebenen der Automatisierung erfolgen und sollte daher im Optimalfall (teil-)automatisiert durchgeführt werden. Modulare Module sind intern wiederum modular aufgebaut und stellen wieder eine modulare Anlage dar. Durch die definierten Schnittstellen zwischen Modulen und Backbone ist die Integration von Modulen verschiedener Hersteller in eine Anlage möglich.

Ein Vorteil dieser Herangehensweise ist, dass Produkte durch die Verwendung von Standardkomponenten und durch eine verbesserte Wiederverwendung von einzelnen Modulen in unterschiedlichen Anlagen schneller in den Markt gebracht werden können. Erreicht eine modulare Anlage oder einzelne Module die Grenzen der Kapazität, kann diese durch ein einfaches Numbering-up, d. h. Ergänzung einer weiteren Anlage, anstelle eines Scale-up (Steigerung der Produktion eines bestehenden Verfahrens) gesteigert werden. Somit muss nicht die Produktivität eines Prozesses als solcher nach oben skaliert werden, sondern es wird lediglich die Anzahl der produzierenden Module erhöht. Durch die dezentrale Struktur steigt insgesamt der Komplexitätsgrad der Automatisierung (vgl. [Die02, Lim94]).

2.1.3 Engineering von automatisierten Systemen

Unter Anlagen-Engineering (dt. Anlagenplanung) werden die Schritte von der ersten Idee einer Anlage über den Bau und die Inbetriebnahme bis hin zu deren Nutzung im normalen Betrieb verstanden [Web14]. Grundlagen des Engineerings sind die Anforderungen des späteren Betreibers. Diese zu sammeln und zu ordnen ist der erste Schritt des Engineerings. Ausgehend von den ermittelten Anforderungen wird die Anlage geplant. An diesem Prozess sind alle notwendigen Fachdisziplinen beteiligt (Verfahrenstechnik, Mess- und Regeltechnik, etc.). Ergebnis der Planung sind verschiedene Artefakte, beispielsweise Modelle der Anlagentopologie (R&I Fließbilder), PLT-Stellenblätter und Steuerungscode in den Sprachen der IEC 61131 (vgl. Kapitel 2.1.1) [HSF+13]. In der Planungsphase wird ebenso die Hardware zur Umsetzung der geplanten Funktionalität ausgewählt. Im Rahmen der Umsetzung wird entsprechend der durchgeführten Planungen die konkrete Anlage errichtet, anschließend getestet und in Betrieb genommen. Nach der erfolgreichen Abnahme durch den Betreiber kann diese den Regelbetrieb aufnehmen. Die anschließenden Wartungs- und Optimierungsarbeiten werden ebenso wie ein möglicher Rückbau als Bestandteil des Engineerings im Anlagenlebenszyklus betrachtet [Web14].

Neben der rein technischen Betrachtung wird unter Engineering auch die Einhaltung von wirtschaftlichen, rechtlichen und organisatorischen Randbedingungen verstanden [Koe85]. Eine automatisierte Anlage muss mit möglichst geringen Aufwänden (d. h. Personal, Material, etc.) errichtet werden. Dabei ist es unerlässlich, die geltenden Vorschriften und Gesetze einzuhalten [Web14]. Die Abwicklung eines Engineering-Projekts ist durch die Beteiligung der vielen Personen eine große Herausforderung. Allerdings muss darüber hinaus auch berücksichtigt werden, dass die Anlage durch die Mitarbeiter der Betreiberorganisation betrieben werden kann.

Engineering-Prozesse

Zur Durchführung von Engineeringprozessen existieren in der Literatur viele Vorschläge und Ansätze. Ein bekannter Ansatz ist der im Folgenden vorgestellte Prozess zur „Abwicklung von PLT-Projekten“ aus dem Arbeitsblatt 35 der NAMUR [NAM03]. Anwendungsbereich des Arbeitsblattes ist die Durchführung von leittechnischen Projekten in der Prozessindustrie. Mit dem Arbeitsblatt wird das Ziel verfolgt, dem wachsenden Kostendruck und der zunehmenden Komplexität der Automatisierung zu begegnen. Dafür wird ein strukturierter Ablauf für die Durchführung von Projekten vorgeschlagen, sowie Empfehlungen für das Qualitäts- und Projektmanagement gemacht.

In Abbildung 2.6 ist der strukturierte Ablauf mit seinen sieben Phasen dargestellt. Innerhalb der Pfeile ist der Name der jeweiligen Phase zu erkennen und unterhalb des Pfeils ist das Zwischenziel der Phase angegeben. Die erste Phase (*Grundlagenermittlung*) besteht aus der Festlegung der Projektziele und einer groben Schätzung der zu erwartenden Kosten auf Basis einer vorliegenden Verfahrensbeschreibung und einer geplanten Anlagenkapazität. Das Ergebnis ist eine durchführbare Anlage. Davon ausgehend werden in der *Vorplanung* das Anlagenkonzept festgelegt und die Kosten genauer kalkuliert. In dieser Phase wird das erste Sicherheitsgespräch durchgeführt und eine Wirtschaftlichkeitsberechnung erstellt. Das Ergebnis ist ein Anlagenkonzept und die dazugehörige Dokumentation

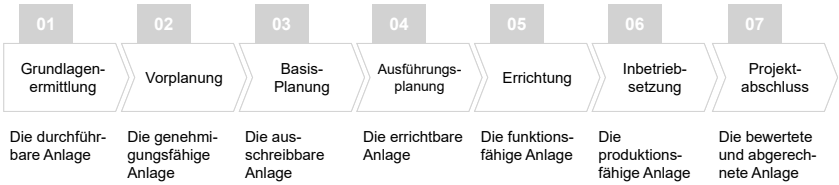


Abbildung 2.6: Darstellung des Prozesses der Abwicklung eines PLT-Projekts nach [NAM03].

(z. B. PLT-Stellenblätter und R&I-Fließbilder). Diese Dokumente werden in der *Basisplanung* verfeinert und die Kostenschätzung fortlaufend aktualisiert. Die Verfeinerung betrifft insbesondere die Beschaffung der verfahrenstechnischen Daten und die Festlegung der technischen Realisierung der leittechnischen Funktionen. In der *Ausführungsplanung* werden die benötigten Geräte und das Leitsystem spezifiziert. Ergebnisse sind u. a. die Stellenfunktionspläne und Montageunterlagen. In der fünften Phase (*Errichtung*) werden die Aufträge ausgestellt, deren Durchführung veranlasst und überwacht. In dieser Phase wird ladefähiger Code für die Leitsysteme programmiert und die Dokumentation erstellt. Dieser Code muss auf Funktionsfähigkeit überprüft und gegebenenfalls müssen Nacharbeiten durchgeführt werden. Abschließend wird die Funktionsfähigkeit dokumentiert. Während der *Inberiebsetzung* wird das Personal ausgebildet, die Anlage in Betrieb genommen und die Dokumentation auf den aktuellen Stand gebracht. In der letzten Phase (*Projektabschluss*) werden der Abschlussbericht und die Abrechnung erstellt.

Der typische Lebenszyklus einer Anlage in der Industrie 3.0 wird in [WGE+17] vorgestellt. Dieser fokussiert die Behandlung von Assets über den Lebenszyklus. So werden in der ersten Phase der *Grobplanung* die Anforderungen und Zusicherungen an ein Gerät festgehalten. In der zweiten Phase wird ausgehend von den Anforderungen ein Gerätetyp ausgewählt. Diese Auswahl erfolgt anhand von Katalogen oder von Typenbibliotheken. Mit den vorliegenden Informationen über die Geräte kann die Detailplanung durchgeführt werden. In diesem Rahmen werden die Konstruktion, die Steuerung und die Infrastruktur geplant. In der vierten Phase wird die Anlage gebaut, getestet und in Betrieb genommen. Anschließend erfolgen die Übergabe und die Aufnahme der Produktion.

Diese beiden Abläufe stehen beispielhaft für die Engineeringprozesse, wie sie aktuell Stand der Technik sind. Ein Überblick über verschiedene Vorgehensmodelle im Engineering ist in [Sch16b] zu finden. Wie auch [Web14] kommt der Autor zu dem Schluss, dass viele Tätigkeiten im Engineering repetitiv sind und daher ein großes Potential für die Nutzung von systematischen Wiederverwendungskonzepten bieten. Die Aufwände für die Planung der PLT-Stellen und der Konfiguration der Software werden auf etwa 30% der Gesamtaufwände geschätzt [NAM03]. Zusätzlich wird die Erzeugung der Software als eine der fehleranfälligsten Projektphasen eingeschätzt. Heutige Automatisierungssystem sind weder fehlerfrei noch 100% testbar [NAM08].

Die geschilderten Abläufe stellen den Optimalfall dar [WGE+17]. In der Praxis ist mit undokumentierten Änderungen an der Hard- und Software zu rechnen. Demzufolge ent-

sprechen wenige Pläne von gebauten Anlagen den jeweiligen Anlagen. Für die einzelnen Projektphasen und Gewerke existieren jeweils unterschiedliche Tools mit proprietären Austauschformaten [VHDB13]. Die heterogene Tool-Landschaft zusammen mit fehlenden Konsistenzchecks wird als die Ursache für Inkonsistenzen und Missverständnisse in Anlagenplänen angesehen [WGE+17].

Die Notwendigkeit, das Wissen und die entwickelten Artefakte aus Projekten in Folgevorhaben nutzbringend einzusetzen, wird in [NAM08] betont. Ausgangspunkt ist die Annahme, dass ein großer Teil der Arbeiten, insbesondere bei der Erstellung der Pläne und der Umsetzung in eine Leitsystemprogrammierung, repetitiver Natur ist. Wenn diese Annahme bejaht wird, so ist es sinnvoll das in den Plänen und der Implementierungen steckende Wissen in nachfolgende und andere Projekte zu übertragen.

Wiederverwendung in der Praxis der Automatisierung

Ein grundsätzlicher Mechanismus der Wiederverwendung in der Automatisierungstechnik ist die Entwicklung von z. B. Funktionsbausteinen und deren Nutzung in mehr als einem Anwendungsfall. Dieser Ansatz wird auch im Bereich der Hardware angewendet, da Pumpen oder Sensoren in mehr als einer Anlage und in großer Stückzahl zum Einsatz kommen [DMG+17].

Für Funktionsbausteine wird in [YGE13, WTPE17] der ACPLT Software Development Process (SDP) vorgestellt. Dieser setzt auf eine strikte Trennung zwischen der Entwicklung der Bausteine und deren Nutzung im Engineering von Lösungen konkreter Aufgaben. Diese Trennung ist nicht nur zeitlich, sondern auch personell zu sehen, da im Rahmen der Entwicklung eher softwaretechnische und im Engineering mehr anwendungsspezifische Fähigkeiten benötigt werden [VHDF+14]. Dieser Prozess ermöglicht es zwar, die Aufwände in der Entwicklung häufiger zu nutzen, allerdings ist die Kenntnis der entwickelten Bausteine eine Grundvoraussetzung. Ist diese nicht gegeben, z. B. durch schlechte Kommunikation in der Organisationseinheit, so kann es zur parallelen Entwicklung funktional gleicher Bausteine kommen.

Ein Ergebnis des Engineeringprozesses sind Funktionsbausteinnetzwerke, die sich aus den vorher entwickelten Bausteinen zusammensetzen [HSF+13]. Diese Netzwerke werden in mehreren Anwendungsfällen verwendet. Im Änderungsfall (z. B. der Fehlerbehebung) müssen diese oft manuell geändert werden bzw. nachgepflegt werden. Da die Pläne oft strukturell identisch sind, sich jedoch in der Parametrierung unterscheiden, ist eine Propagierung der Änderungen nicht durch ein *Kopieren und Einfügen* zu erreichen.

Es gibt unterschiedliche Hersteller, deren Blöcke nicht kompatibel sind. Selbst bei der Verwendung von Standardbausteinen kann es sein, dass deren Interface nicht gleich ist, was zu Problemen bei der Übertragung einer Änderung auf das System eines anderen Herstellers führt. Bei der Verwendung von selbst entwickelten Funktionsbausteinen und Funktionsbausteinnetzen sind die Lösungen nur eingeschränkt übertragbar. Dies gilt auch für die Migration von einer Leitsystemversion auf eine andere. Ein Grund für die fehlende Übertragbarkeit ist die unterschiedliche Umsetzung von ST-Befehlen in den Systemen der einzelnen Hersteller. Dies macht eine Übertragung von Applikationen zwischen den Herstellern schwierig.

Eine weitere Art der Wiederverwendung ist die Nutzung von einheitlichen Strukturen. Dies gilt u. a. für die Strukturen von Reglern. Ein Beispiel ist die Verwendung von Kaskaden oder von Störgrößenaufschaltungen [ASE08]. Durch dieses Vorgehen kann, eine richtige Auswahl der Struktur vorausgesetzt, die Problemstellung auf die korrekte Parametrierung der Bausteine reduziert werden.

2.1.4 Quo vadis Automatisierungstechnik? – Ein Ausblick im Zeitalter von Industrie 4.0

Neben dem aktuellen Stand der Automatisierungstechnik in der Praxis, wird im Folgenden die nahe Zukunft betrachtet. Allgemein herrscht die Überzeugung, dass die Automatisierung und mit ihr die gesamte produzierende Industrie vor größeren Veränderungen steht [SAG+17]. Ein Anzeichen dieser Entwicklung sind Projekte wie die *Industrie 4.0* Initiative in Deutschland und das von der Object Management Group (OMG) in den Vereinigten Staaten ins Leben gerufene *Industrial Internet Consortium*. Diese beiden Initiativen sind jeweils eine Sammlung von Vorhaben und Projekten, die von unterschiedlichsten Beteiligten vorangetrieben werden. Ihnen gemeinsam ist jedoch das übergeordnete Ziel, die Digitalisierung und Vernetzung der industriellen Produktion voranzutreiben. Damit ist nicht nur die Vernetzung innerhalb der Betriebe und Unternehmen gemeint, sondern vor allem die Vernetzung über die Grenzen von Unternehmen hinweg. Ziel ist es, ein integriertes Wertschöpfungsnetzwerk zu schaffen, das mindestens teil-automatisiert Daten über Produkte und Aufträge zwischen verschiedenen Teilnehmern austauscht. Damit soll insbesondere an Hochlohnstandorten eine effizientere Produktion ermöglicht werden.

Zusätzlich zu den beschriebenen Zielen soll die Produktion insgesamt flexibler gestaltet werden. In diesem Kontext ist eine Produktion mit der Losgröße eins die ultimative Benchmark [SAG+17]. Ein Schritt zur Erreichung dieses Ziels ist die Realisierung einer flexiblen und wandelbaren Produktion in technischen Betrieben. Neben der Entwicklung von neuer Infrastruktur, die eine variable Anordnung der Produktionsmittel ermöglicht, muss auch die Automatisierungslösung in die Lage versetzt werden, der Änderung des Produktionsprozesses zu folgen. Eine Möglichkeit dies umzusetzen ist die Programmierung zu flexibilisieren [VHDF+14]. Dies kann beispielsweise durch die Verwendung von Laufzeitsystemen, die zur Laufzeit rekonfiguriert werden können, erfolgen. Durch das Halten von Engineeringdaten innerhalb der Laufzeitumgebung kann die Automatisierungslösung durch eine automatisierte Transformation aus einer Problembeschreibung und den Randbedingungen erzeugt werden. Um die Lücke zwischen der geplanten Änderung und der echten Adaption eines neuen Produktionsprozesses oder neuer Randbedingungen zu schließen, müssen die Systeme in der Lage sein, ihre Umgebung zu erkennen und die Funktionalität der angeschlossenen Geräte zu erkunden [VHDB13]. Zur Realisierung der Erkundbarkeit bietet sich OPC UA als Kommunikationsprotokoll an. Neben der verbesserten Kommunikation spezifiziert OPC UA ein Metamodell für die objektorientierte Datenmodellierung. Ausgehend davon können Datenmodelle für die unterschiedlichsten Anwendungsfälle entwickelt werden (z. B. Verwaltungsschalen [PE17]).

Flexibilität von Produktionssystemen bedeutet die vorher geplante reversible Anpassung an neue Gegebenheiten [Löf11]. Charakteristisch für flexible Produktionssysteme ist der

begrenzte Handlungsspielraum, innerhalb dessen sich diese bewegen können. Wandelbare Systeme können diesen engen Korridor verlassen und im Vorhinein nicht geplante Veränderungen durchführen [Löf11].

Ein Weg zur Umsetzung der Adaption einer neuen Aufgabe durch die Automatisierungslösung ist die Nutzung einer wandelbaren Struktur [WE15a]. Wandelbar bedeutet in diesem Zusammenhang, dass sich die Struktur ändern können muss. Dies kann nötig sein, wenn sich die Struktur der Anlage ändert oder neue Komponenten in die Automatisierungslösung eingebracht werden. Dies kann durch eine dienstbasierte Architektur und die Verwendung von Nachrichten für das Aufrufen der Dienste realisiert werden [EE13]. Dabei bieten die Prozessführungskomponenten die von ihnen angebotene Funktionalität über ein Dienstsystem an. Diese Dienste können von Dienstnutzern, d. h. anderen Prozessführungskomponenten oder Bedienern, gefunden und genutzt werden. Für die Nutzung in der Automatisierungstechnik sind Ressourcen orientierte Architekturen (ROA) geeignet [WE17]. Diese Architekturen sind reduzierte Dienstsysteme, die anstelle von Diensten mit einem großen Interface und beliebiger Funktionalität auf atomare Dienste mit einer begrenzten Funktionalität setzen. Die Komplexität eines umfassenden Dienstes kann gegen komplexere Aufrufe von atomaren Diensten getauscht werden. Microservices [BFK+17] und die Dienste für die Interaktion von Komponenten [IEC05] sind Beispiele dafür.

Ein weiterer Gegenstand der Betrachtung ist die Anbindung der Automatisierungslösung an die Cloud [SCZ+16]. In diesem Kontext wird *Edge Computing*, d. h. die Datenverarbeitung an der Grenze zur Cloud diskutiert. Für das Edge Computing werden die Ressourcen von Geräten verwendet, die nahe am Prozess stehen. Aufgaben sind beispielsweise die Datenverarbeitung oder die Verteilung von Anfragen [SCZ+16]. Vorteile gegenüber dem reinen Cloudcomputing sind die bessere Anbindung an die Produzenten von Daten und die daraus resultierenden höheren Bandbreiten sowie geringere Antwortzeiten.

Durch die zunehmende Verwendung von smarten Geräten und deren Vernetzung stehen in Zukunft immer mehr Daten über Prozesse und Geräte zur Verfügung. Um diese einheitlich zugänglich zu machen, sind ein standardisiertes Interface und eine vereinheitlichte Datenmodellierung erforderlich. Mit der Verwaltungsschale wird ein solches Interface und eine objektorientierte Datenmodellierung vorgeschlagen [PE17]. Dabei besteht die Verwaltungsschale aus verschiedenen Elementen, die z. B. durch einen OPC UA-Server von außen zugänglich gemacht werden können. Eine Anwendung für die neuen Daten sind Applikationen aus dem Bereich des *Predictiv Maintenance*. Dabei wird versucht, aus den Daten eine Veränderung des Gerätezustands herauszulesen und einen möglichen Ausfall vorherzusagen. Eine Vernetzung der Systeme über die Unternehmensgrenzen hinweg ermöglicht es, dass der Hersteller Zugang zu den Daten seiner verbauten Geräte erhält und diese auswerten kann. So können neue Dienstleistungen realisiert werden.

In [WGE+17] wird ein überarbeiteter Lebenszyklus für die Anlagenentwicklung vorgestellt (vgl. Abbildung 2.7). Der Prozess ist im Wesentlichen deckungsgleich zu dem vorgestellten aus dem *NAMUR Arbeitsblatt 35*. Er besteht aus den Schritten *Grobplanung*, der anschließenden Geräteauswahl aus dem Katalog von Typen, dem Aufbau des Instanz-Modells der Anlage und dem abschließenden Aufbau der Anlage.

Der Fokus liegt auf der Verwendung von neuen digitalen Modellen in den verschiedenen Schritten der Planung einer Anlage. In Kombination mit einer verbesserten Werkzeugkette

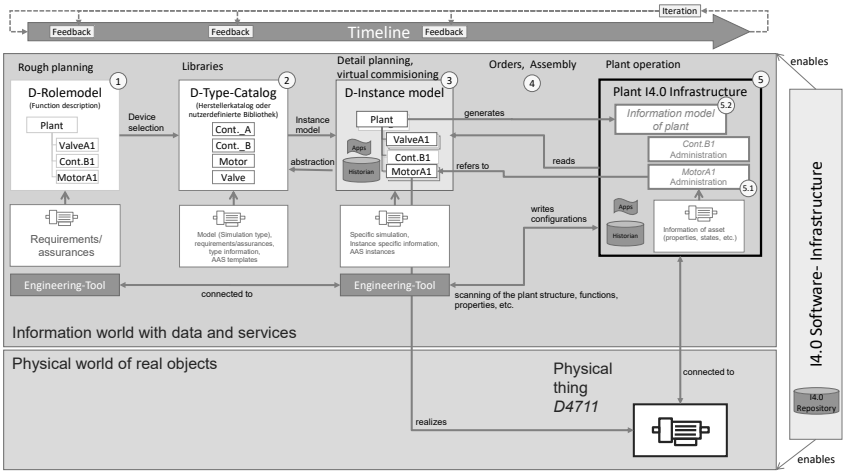


Abbildung 2.7: Darstellung der Anlagenentwicklung im Kontext von Industrie 4.0 nach [WGE+17]

soll diese Entwicklung zu geringen Aufwänden im Engineering führen. Die verbesserten Werkzeugketten bestehen aus passenden Schnittstellen zwischen den unterschiedlichen Systemen, die eine automatisierte Datenübertragung ermöglichen und einer durchgängigen Betrachtung von Assets über deren gesamten Lebenszyklus hinweg ermöglichen. Eine angestrebte Verbesserung ist die Verkleinerung der Lücke zwischen den Planungs- bzw. Dokumentationsunterlagen und der Anlage selbst. Dies soll durch eine Rückwärts-Propagierung von Änderungen an Planungsmodellen z. B. der zweiten und dritten Phase in die erste oder zweite Phase erfolgen. Somit sind alle Modelle auf dem Stand der Anlage.

2.2 Komponentenbasierte Architekturen

Im folgenden Abschnitt werden die Grundgedanken zu komponentenbasierten Architekturen vorgestellt und erläutert. Zunächst wird der Begriff *Komponente* eingeführt und eine Definition für die vorliegende Arbeit gegeben. Komponenten werden in Systemen und Architekturen zusammengefasst. Deren Aufbau und Vorteile widmet sich der nachfolgende Abschnitt. Anschließend werden die Verwendung von komponentenbasierten Architekturen in der Automatisierungstechnik vorgestellt.

2.2.1 Der Komponentenbegriff

Die Verwendung von Komponenten sind ein sehr grundsätzlicher Ansatz zur Betrachtung eines Systems, das aus einzelnen Teilen, d. h. Komponenten zusammengesetzt ist. Im Duden wird Komponente als „Bestandteil bzw. Element eines Ganzen“ definiert. Nach dieser

Definition sind Komponenten Elemente eines übergeordneten Systems. Beispiele für eine derartige Betrachtung sind die Komponenten einer Stoffmischung oder die Komponenten in einem Computer. Der Begriff Komponente stellt sich als generisch dar und abhängig vom Gegenstand der Betrachtung können darunter sehr verschiedene Dinge verstanden werden.

In der DIN SPEC 40912 [DIN14] wird daher die *technische Komponente* als Spezialisierung der allgemeinen Komponente eingeführt. Unter dem Begriff werden sowohl Hard- als auch Softwarekomponenten subsumiert.

Definition 1 (Technische Komponente)**.** *Vorgefertigte, in sich strukturierte und unabhängig hantierbare Einheit zur Realisierung einer konkreten Rolle in einem System [DIN14].*

Die Definition der technischen Komponente beinhaltet vier zusätzliche Kernaspekte: sie ist vorgefertigt, strukturiert, unabhängig hantierbar und realisiert eine konkrete Rolle. Vorgefertigt bedeutet, dass die technische Komponente schon vorliegt, wenn das System aufgebaut wird. Die technische Komponente hat einen internen Aufbau, der nach einem Plan aufgebaut ist. Die interne Struktur muss nicht zu jedem Zeitpunkt transparent sein, d. h., eine technische Komponente kann nach außen als Black-Box erscheinen (vgl. [OMG15]). Technische Komponenten können einem System hinzugefügt oder entnommen werden, beispielsweise, wenn sie getauscht werden. Hierdurch erfolgt eine Abgrenzung zur allgemeinen Komponente, da diese nicht einzeln hantierbar sein muss. Die Komponenten des Komponentenklebers bilden nach der Vermischung eine Einheit und die Komponenten können nicht mehr getrennt behandelt werden. Diese Eigenschaft einer Komponente stellt besondere Anforderungen an das Interface der Komponente und der Umgebung, in welche die Komponente eingebaut wird. Die beiden Interfaces müssen zueinander kompatibel sein. Zudem muss die Umgebung die Möglichkeiten bereitstellen, die Komponente zu hantieren. Im Folgenden werden technische Komponenten als Komponenten bezeichnet. Der letzte Aspekt der Definition von Komponenten ist, dass technische Komponenten eine konkrete Rolle in einem System erfüllen. Die Grundlagen dieser Systembeschreibung werden nachfolgend erläutert.

Rollen und Realisierungseinheiten – Das Rollenmodell

Das Rollenmodell ist eine Systembeschreibung, die im Engineering von Anlagen verbreitet ist [WGE+17]. Es beschreibt das System als Kombination aus Rollen und Realisierungseinheiten (vgl. Abbildung 2.8) [DIN14]. Rollen beinhalten auf der einen Seite die Anforderungen an Realisierungseinheiten und auf der anderen Seite Zusicherungen an die umgebenden Systemelemente und das Gesamtsystem. Realisierungseinheiten realisieren diese Rollen und werden Bestandteil des Systems.

Die Trennung in Rolle und Realisierungseinheit stellt einen fundamentalen Bestandteil der Anlagenplanung dar und ist, wenn auch nicht immer explizit beschrieben, zumindest implizit vorhanden [DMG+17]. Wird zum Beispiel der Temperatursensor einer Anlage getauscht, so wird der alte Sensor der Anlage entnommen und damit die Verbindung zwischen der Rolle „Temperatur messen“ und dem Sensor als Realisierungseinheit getrennt. Die Rolle, d. h. die Anforderung, dass eine Temperatur unter den gegebenen Randbedingungen gemes-

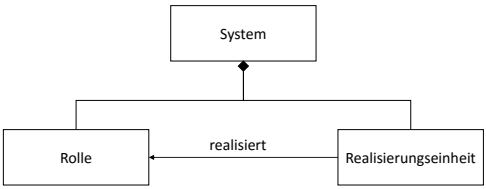


Abbildung 2.8: Darstellung des Rollenmodells zur Systembeschreibung [DIN14]

sen werden soll, existiert weiterhin. Sie wird durch den neu eingebauten Sensor realisiert. Dieser ist dann die neue Realisierungseinheit.

Bei der Betrachtung des Lebenszyklus einer Anlage wird der Nutzen des Rollenmodells deutlich. In [DMG+17] beschreiben die Autoren die Verwendung von Rollenmodellen im Engineering. Hierbei wird die Anlage zunächst abstrakt (z. B. in einem Rohrleitungs- und Instrumentierungsfließbild [IEC16]) entworfen. Dabei wird unter Verwendung der Rollen eine Anlagenstruktur entworfen, ohne die konkreten Gerätetypen zu benennen, die in der Anlage verbaut werden sollen. Diese dokumentieren die Anforderungen an die Geräte und deren Aufgabe in der Anlage. Unter Verwendung dieser Anforderungen können geeignete Typen von Realisierungseinheiten gesucht werden. Ausgehend von dem Ergebnis dieser Suche werden die entsprechenden Realisierungseinheiten festgelegt [DF04, Epp08]. Der Plan für den funktionalen Aufbau (Rollenmodell) einer Anlage ist damit losgelöst von den tatsächlichen Geräten in der Anlage. Diese Trennung wird bei der Verwendung von Verwaltungsschalen für Geräte und der damit verbundenen Nutzung von Lebenszyklusinformationen genutzt [WGE+17].

Der Aufbau von Komponenten

Aus der Definition von technischen Komponenten geht hervor, dass diese in sich strukturiert sein müssen. Ein Vorschlag für die Beschreibung dieses inneren Aufbaus von Softwarekomponenten wird in [Ens01] vorgestellt. Dabei werden zwei Arten von Komponenten unterschieden: primitive und komplexe Komponenten. Primitive Komponenten sind atomar, d. h., sie können in keine weiteren Bestandteile (Unterkomponenten) zerlegt werden. Im Gegensatz dazu setzen sich komplexe Komponenten aus Unterkomponenten zusammen. Die Unterteilung ist analog zu der Einteilung in der IEC 61499 [IEC05]. Für die Beschreibung von Komponenten als Black-Box werden *Kapseln* eingeführt [Ens01]. Diese beschreiben das Interface einer oder mehrerer Komponenten. Das beschriebene Interface ist eine Mindestanforderung, d. h., die Komponenten können über ein größeres Interface verfügen und sind trotzdem kompatibel. In [Ens01] werden Kapseln als Beschreibung für Komponenten vorgesehen, die „Gemeinsamkeiten in Struktur und Verhalten aufweisen“. Kapseln können als Rollenbeschreibung von Funktionsbausteinen aufgefasst und im zugehörigen Engineeringprozess für den Aufbau von komplexen Komponenten verwendet werden. Im weiteren Verlauf werden die Kapseln durch Funktionsbausteine implementiert.

Eine sehr prominente Beschreibung für den Aufbau von Softwarekomponenten ist in der

Spezifikation der Unified Modeling Language (UML) enthalten [OMG15]. Darin wird eine Komponente als modulare Einheit mit einem wohldefinierten Interface, die innerhalb ihrer Umgebung austauschbar ist, definiert. Diese Definition ist für Softwarekomponenten deckungsgleich mit der vorgestellten Definition der technischen Komponenten. Der Fokus der Komponente aus Sicht der UML liegt auf der Kapselung des Zustands und des Verhaltens der Komponente. Darüber hinaus sind Komponenten austauschbare Elemente, wenn ihre Funktionalität und ihr Interface kompatibel sind. In Abbildung 2.9 ist der strukturierte Aufbau von Komponenten vereinfacht dargestellt. Es ist zu erkennen, dass Komponenten von einer Klasse abgeleitet werden und aus Realisierungen, PackageableElements und Interfaces bestehen können. Das Interface unterteilt sich in benötigte (*required*) und bereit gestellte (*provided*) Interfaceelemente. Ein weiterführender Überblick zu Softwarekomponenten ist in [Die02] zu finden.

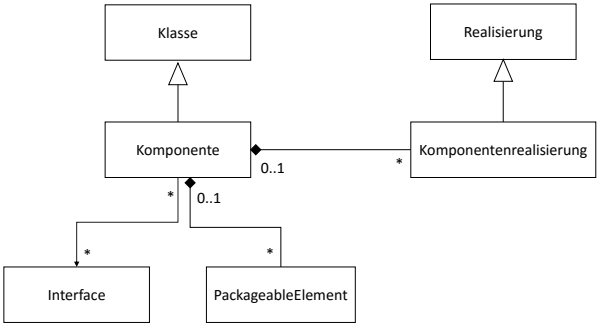


Abbildung 2.9: Komponentenaufbau nach [OMG15]

Eine andere Art der Komponenten sind Hardwarekomponenten, wie z. B. Feldgeräte (vgl. Kapitel 2.1.1) oder Package Units (vgl. Kapitel 2.1.2). Der innere Aufbau dieser Hardwarekomponenten ist für die Betrachtung in einem größeren System (z. B. einer Anlage) selten relevant. Daher werden diese Elemente bei der Beschreibung von Anlagen (z. B. in einem Rohrleitungs- und Instrumentierungsfließbild [IEC16]) als Black-Boxen dargestellt. Aus Sicht der Systembeschreibung sind nur die Funktionalität und das Interface (z. B. Art des Anschlusses oder benötigte Spannungsversorgung) relevant. Analog dazu ist die Definition des Module Type Package (MTP) der Interessengemeinschaft Automatisierungstechnik der Prozessindustrie (NAMUR) zu verstehen, in dessen Rahmen die Schnittstellen für Module definiert werden. Für jedes Modul muss die Mensch-Maschine-Schnittstelle und eine Schnittstelle für das Steuern und Überwachen beschrieben sein [BHH+16]. Für das Steuern und Überwachen stellt das Modul seinen internen Zustand nach außen dar und bietet (Prozessführungs-)Dienste an, die von einer überlagerten Steuerung genutzt werden. Der interne Aufbau des Moduls ist für den Nutzer im Sinne einer Service orientierte Architektur (SOA) nicht relevant und kann durch verschiedene Hardwareaufbauten realisiert werden.

Eigenschaften von Komponenten

Eigenschaften von Softwarekomponenten, die auch für technische Komponenten gelten, sind in [Sam97] vorgestellt worden. Die wesentlichen Eigenschaften sind:

- **Geschlossenheit**
  Unter Geschlossenheit ist zu verstehen, dass eine Komponente wiederverwendet werden kann, ohne dass dazu andere Komponenten oder Hilfsmittel nötig sind. Sind für die Verwendung einer Funktion andere Funktionen erforderlich, so müssen diese alle in einer Komponente enthalten sein. Eine Klasse aus der Softwaretechnik ist im Allgemeinen alleine keine Komponente, da aufgrund von Vererbungsstrukturen für deren Erzeugung weitere Klassen vorhanden sein müssen.
- **Identifizierbarkeit**
  Komponenten müssen eindeutig identifizierbar sein. Zusätzlich zu der reinen Namensgebung ist darunter auch die Auffindbarkeit zu verstehen.
- **Klarheit der Schnittstelle**
  Bei der Verwendung von Softwarekomponenten steht die Wiederverwendung im Vordergrund. Daher muss sich die Schnittstelle auf den für die Wiederverwendung relevanten Umfang beschränken und alles Weitere verbergen. Im Allgemeinen wird unter den Schnittstellen die Signatur der angebotenen Funktionen verstanden [Die02].
- **Dokumentation**
  Die Dokumentation von Komponenten ist für deren Verwendung und insbesondere für die Wiederverwendung wichtig. Die am besten für die Wiederverwendung verwendbare Komponente ist nutzlos, wenn es keine geeignete Dokumentation gibt.
- **Wiederverwendungsstatus**
  Der Wiederverwendungsstatus gibt an, wer die Komponente besitzt, wer für die Wartung verantwortlich ist und wer kontaktiert werden kann, wenn es Probleme mit der Komponente gibt.

In [Die02] wird zwischen zwei Arten von Komponenten unterschieden: den horizontalen und den vertikalen Komponenten. Horizontale Komponenten sind unabhängig von einem konkreten Anwendungsgebiet bzw. einer Domäne. Beispiele sind Komponenten für Benutzerschnittstellen oder das Datenmanagement. Im Gegensatz dazu stellen vertikale Komponenten Funktionalitäten für einen konkreten Bereich bereit. Die Prozessführungskomponenten sind dafür ein Beispiel.

Abbildung 2.10 zeigt das Zusammenspiel aus *Komponentenentwicklung*, *Komponentenmanagement* und *Lösungsentwicklung*. Im Kern dieser drei Aufgaben steht das *Repository*, in dem die Komponenten gesammelt und verwaltet werden. Im Rahmen der Komponentenentwicklung werden neue Komponenten dem Repository hinzugefügt, die im Nachgang von den Entwicklern bearbeitet und verbessert werden können. Die Verwaltung der Komponenten und ihre Bereitstellung für die Lösungsentwicklung ist Gegenstand des Komponentenmanagements. Ausgehend von den entwickelten Komponenten werden in der Lösungsentwicklung Lösungen für die jeweiligen Problemstellungen realisiert. In [HC01] wird betont, dass ein zentrales Repository für die Durchführung der Prozesse wichtig ist, damit die verfügbaren Komponenten allen Beteiligten bekannt sind und von diesen genutzt

werden können.

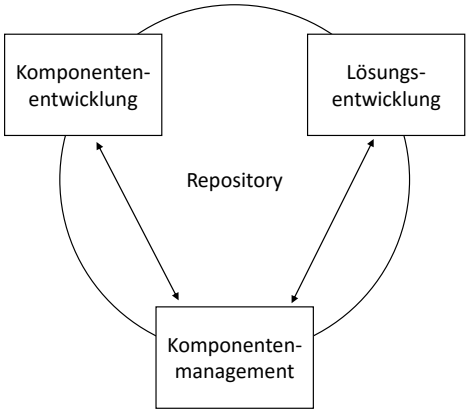


Abbildung 2.10: Das Repository im Zentrum der Komponentenverwendung nach [HC01]

Komponenten als Grundlage für die Wiederverwendung

Einheiten, seien sie physisch oder nicht physisch, werden erst durch das Hinzufügen zu einem Komponentensystem zu einer Komponente. Übertragen bedeutet das, dass der Sensor im Lager oder die in der Cloud gespeicherte Softwareapplikation keine Komponenten im engeren Sinne sind. Im allgemeinen Sprachgebrauch werden jedoch auch Einheiten, die prinzipiell als Komponenten verwendet werden können, als Komponenten bezeichnet, auch wenn sie aktuell keinem Komponentensystem zugeordnet sind [DIN14].

Nach [Kru04] sind komponentenbasierte Architekturen eine Lösung für grundlegende Probleme im Bereich der Softwareentwicklung, da sie folgende Vorteile haben:

- Komponenten erleichtern das Design von widerstandsfähigen Architekturen.
- Modularität ermöglicht eine klare Trennung der Zuständigkeiten von Systemelementen.
- Wiederverwendung wird durch standardisierte Frameworks und verfügbare Komponenten erleichtert.
- Komponenten stellen eine natürliche Basis für das Konfigurationsmanagement dar.
- Tools für die visuelle Modellierung stellen eine Automation für das komponentenbasierte Entwickeln dar.

Dementsprechend sind komponentenbasierte Systeme in vielen Bereichen zumindest implizit verbreitet. Wie gezeigt wurde, existieren Definitionen von Komponenten sowohl im Hard- als auch im Softwarebereich. Im Folgenden werden beispielhaft einige komponentenbasierte Architekturen aus der Automatisierungstechnik vorgestellt.

2.2.2 Komponentenbasierte Architekturen in der Automatisierungstechnik

Beispiele für komponentenbasierte Architekturen in der Automatisierungstechnik sind Funktionsbausteine aus der IEC 61131 [IEC14b] und IEC 61499 [IEC05], Package Units, die Betrachtung von Sensoren und Aktoren in Anlagen und Rohrleitungs- und Instrumenten-Fließbilder. Die Funktionsbausteine und die Hardwarekomponenten sind klassische Beispiele für komponentenbasierte Architekturen. Aber auch die Elemente von Fließbildern können als Komponenten aufgefasst werden, da sie in ihrer Umgebung einzeln handhabbar sind, einen konkreten Zweck haben und ein definiertes Interface bereitstellen.

Softwarekomponenten

Für den Aufbau des Softwareteils der Automatisierungslösung (System für die Automatisierung z. B. einer Anlage, vgl. [VHDF+14]) werden in der Regel die Programmiersprachen der IEC 61131 verwendet [JT00]. Zur Strukturierung hat sich die Verwendung von Funktionsbausteinen in Funktionsbausteinnetzwerken etabliert. Dabei werden einzelne Aufgaben in einem Funktionsbaustein gekapselt. Dieser verfügt über ein definiertes Interface, sogenannte Ports, und kann innerhalb des Automatisierungssystems instanziiert, gelöscht und manipuliert werden. Der rückwirkungsfreie Austausch von Daten wird durch Signalverbindungen zwischen den Ports von Bausteinen realisiert. Als Erweiterung der klassischen Bausteintechnik im Hinblick auf Verteilbarkeit von Lösungen und der Ausführungssemantik ist die IEC 61499 entwickelt worden. Als Ergänzung zu den klassischen Ports sind Eventports zur Abbildung eines Eventflusses eingeführt worden. Funktionsbausteine können zu aggregierten Funktionsbausteinen zusammengefasst werden. Durch die Verwendung von globalen Variablen in einem Funktionsbaustein ist dieser keine Komponente im Sinne der vorgestellten Definition. Diese Bausteine sind nicht abgegrenzt und können nicht in Umgebungen ohne die globale Variable eingesetzt werden. Softwarekomponenten eignen sich besonders gut für die Wiederverwendung, da die Grenzkosten für die Nutzung einer weiteren Instanz vernachlässigbar gering sind. Eine Ausnahme sind Lizenzmodelle, die eine Zahlung an den Lizenzgeber pro eingesetzter Instanz vorsehen.

Hardwarekomponenten

Wie bereits angesprochen, werden auch Hardwareteile als Komponenten in einer Anlage angesehen. Sie sind Bestandteil der Automatisierungslösung. Solche Komponenten sind Sensoren und Aktoren wie z. B. Pumpen und Temperatursensoren. Auch diese Hardwarekomponenten verfügen über ein definiertes Interface, das sie gegenüber ihrer Umgebung

austauschbar macht. Beispiele dafür sind ein Anschluss über Zweidraht, d. h. 4 bis 20 mA, oder Profibus.

Eine Form der Aggregation von Hardwarekomponenten wird bei der Verwendung von Package Units umgesetzt [NAM96]. Dabei werden Module einer Anlage als komplette Einheit bereitgestellt. Ein Beispiel dafür sind Verpackungsmaschinen. Diese können über eine eigene Automatisierungslösung verfügen, die sich entweder in ein übergeordnetes Leitsystem einfügt oder eigenständig betrieben wird. Die Wiederverwendung von Hardwarekomponenten ist analog zu der Klasse-Instanz Beziehung in Softwaresystemen. Ein Pumpentyp beispielsweise kann mehrfach produziert und anschließend in verschiedenen Anlagen eingesetzt werden. Auch ein Tausch von Pumpen unterschiedlichen Typs ist bei gleichem Interface und kompatibler Funktionalität ohne Rückwirkung auf die Umgebung möglich.

Komponenten im Engineering

Bei der Planung einer Prozessanlage entstehen Diagramme, die den geplanten Aufbau der Anlage bestehend aus den Rohren und Instrumenten (R&I- Fließbilder) darstellen [HSF+13, PE94]. Sie sind das Ergebnis der verfahrenstechnischen Anlagenplanung und bilden die Grundlage für die Planung der Automatisierungslösung. Die einzelnen Bestandteile der Diagramme (z. B. Messstellen und Aktoren) sind mit den jeweiligen Anforderungen an sie und den Verbindungen zwischen ihnen dargestellt und können als Komponenten aufgefasst werden. Die Diagramme haben zwei Aufgaben: Auf der einen Seite stellen sie ein Rollenmodell für die zu bauende Anlage und ihre Komponenten dar. Auf der anderen Seite sind sie selbst Modelle, die sich aus einzelnen Komponenten zusammensetzen. Als elektronisches Austauschformat wird mit PandIX eine CAEX Bibliothek auf Basis der IEC 62424 [IEC16] vorgestellt.

Verwendung von Komponenten in Laufzeitsystemen

Durch die zunehmende Verwendung von OPC UA [IEC10, GPP16] und der damit verbundenen Modellierung von Daten und Zusammenhängen in objektorientierten Informationsmodellen, wird die Verwendung von komponentenorientierten Architekturen zunehmen [SAG+17, WTPE17]. In diesen Informationsmodellen werden viele Informationen gespeichert und zugänglich gemacht. Daher ist davon auszugehen, dass diese zunehmend größer und komplexer werden. So muss es ein Anliegen sein, den Aufwand für die Erstellung und Pflege dieser Modelle so gering wie möglich zu halten.

Eine Möglichkeit, diese Modelle zu speichern, sind Laufzeitumgebungen (vgl. Kapitel 2.1.1). Abhängig von der Funktionalität der Laufzeitumgebung kann mit ihr die Abarbeitung von aktiven Komponenten wie Diensterbringern oder Steuerungslogiken durchgeführt werden. Im Kontext der geplanten Umsetzung von wandelbaren Fabriken ist nicht nur die Erkundung von Funktionalitäten zur Laufzeit erforderlich, sondern darüber hinaus auch die Möglichkeit zu deren Modifikation, um auf neue Situationen und Randbedingungen reagieren zu können.

Zur Umsetzung der flexiblen Prozessführung, wie sie durch die Flexibilisierung der Produktion in Industrie 4.0 gefordert ist [SAG+17, VHDF+14], rücken zunehmend Laufzeitumge-

bungen in den Fokus. Diese müssen zumindest teilweise echtzeitfähig sein und flexibel an die Umgebung angebunden werden können. Für eine flexible Erkundung der Funktionalität und Zusammenschaltung von mehreren Laufzeitumgebungen zur Laufzeit ist die Reflexion nützlich. Durch diesen Mechanismus wird der aktuelle Zustand der Laufzeitumgebungen sowie die in ihr enthaltenen Objekte und deren Zustand von außen zugänglich. Die vorgestellten Prozessführungskomponenten können so flexibel realisiert und in wandlungsfähigen Strukturen angeordnet werden [WTPE17]. Ebenso bieten sich Laufzeitumgebungen für Implementierung von Verwaltungsschalen an, die ebenfalls als Komponenten anzusehen sind [PE17].

2.3 Zwischenfazit

In diesem Kapitel wurden die in der Automatisierungstechnik verwendeten Soft- und Hardwaresysteme vorgestellt. Der Fokus lag dabei auf der Prozessindustrie als Anwendungsgebiet. In einem zweiten Teil wurde der Begriff der technischen Komponente eingeführt und existierenden Definitionen von Komponenten gegenübergestellt. Es zeigt sich, dass die automatisierungstechnischen Soft- und Hardwaresysteme als Komponentensysteme aufgefasst werden können. Aufgrund des Fokus der Automatisierungstechnik ist es sinnvoll, Soft- und Hardware nicht getrennt zu betrachten, sondern als hybrides System zu verstehen. Dies gilt insbesondere deshalb, da sich die Soft- und Hardware der Systeme gegenseitig beeinflussen können.

Im Engineeringsprozess werden viele Tätigkeiten häufig und repetitiv durchgeführt, um ähnliche Problemstellungen zu lösen. Hier führt eine verbesserte Wiederverwendung zu geringeren Aufwänden. Zusätzlich ist es durch die verschiedenen Personen, die an der Umsetzung beteiligt sind, schwierig, bestehende (Teil-)Lösungen wiederzuverwenden, da deren Bekanntheit nicht hoch genug ist.

Innerhalb der einzelnen Phasen des Engineeringprozesses werden Komponenten bereits entlang der Grenzen von Gewerken wiederverwendet. Es werden sowohl Soft- als auch Hardwarekomponenten standardisiert. Die Verwendung ist jedoch nicht flächendeckend gegeben. Zusätzlich muss die Standardisierung über die Grenzen der Gewerke sinnvoll und konsistent vorgenommen werden. Selbst wenn innerhalb der Gewerke Komponenten standardisiert sind, müssen diese in hybriden Systemen über die Grenzen der Gewerke zueinander kompatibel sein. Die Wiederverwendung von Komponentensystemen ist bisher bestenfalls in Ansätzen gängige Praxis.

Die relevante Information zur Lösung eines Problems bzw. der Mehrwert einer Implementierung befindet sich in den komponentenorientierten Architekturen der Automatisierungstechnik nicht nur in den Komponenten selbst. Zu einem großen, je nach Anwendung auch überwiegenden, Teil sind die Informationen im Zusammenspiel der Komponenten enthalten. In diese Verknüpfung der Komponenten bzw. in den Aufbau der Komponentennetzwerke wird viel Aufwand investiert. Da die Systeme (z. B. Leitsysteme) ähnlich aufgebaut sind, bzw. funktionell die gleichen Funktionsbausteintypen verwendet werden, besteht in der Übertragung dieser Komponenten ein enormes Potenzial. Man kann diese als Implementierungs-unabhängigen Teil der Lösung von Automatisierungsaufgaben be-

zeichnen.

Durch die Grenzen zwischen den Anwendungen findet in der Praxis keine Wiederverwendung oberhalb der Komponenten statt. Die Dokumentation der Anlagen und der Automatisierungslösungen entspricht nicht immer dem aktuellen Zustand der Systeme.

Es wird daher ein Konzept benötigt, das die Wiederverwendung von komponentenbasierten (Teil-)Lösungen ermöglicht und unterstützt. Dazu muss ein Weg entwickelt werden, diese Teillösungen und die Abhängigkeiten zwischen ihnen zu beschreiben. Der Ansatz muss in die bestehende Architektur eingefügt werden können. Im Rahmen des Konzepts muss sichergestellt sein, dass es sich in die bestehenden Prozesse des Engineerings einfügt und auf zukünftige Entwicklungen der Automatisierungstechnik vorbereitet ist. Das bedeutet, die Anwendung in verteilten Systemen und die Abbildung von dienstbasierten Architekturen müssen unterstützt werden.

3 Anforderungen an das Konzept im Kontext der Automatisierungstechnik

In diesem Kapitel werden die Anforderungen an ein Konzept zur Unterstützung der Wiederverwendbarkeit in komponentenbasierten Architekturen spezifiziert. In einem ersten Schritt werden die Besonderheiten der Automatisierungstechnik vorgestellt. Diese bilden die Grundlage für die anschließend beschriebenen funktionalen und nicht-funktionalen Anforderungen. Diese Anforderungen beschreiben die geforderte Funktionalität und die Art und Weise, wie sich das Konzept in die bestehenden Konzepte und Vorgehensweisen der Automatisierungstechnik einfügen muss.

3.1 Besonderheiten in der Automatisierungstechnik

Das Anwendungsfeld der Prozessautomation stellt besondere Anforderungen an die verwendeten Automatisierungslösungen. Da die vorliegende Arbeit ein Problem dieser Domäne adressiert, ist eine Betrachtung ihrer besonderen Eigenschaften erforderlich. Nur deren Berücksichtigung ermöglicht die Entwicklung einer passgenauen Lösung.

Konsequenzen bei Störungen des Betriebs: In der Automatisierungstechnik hat die Sicherheit einen sehr hohen Stellenwert, da die Auswirkungen einer Störung, insbesondere in der Prozessautomation, sehr hoch sein können [FA09]. Diese reichen von wirtschaftlichen Konsequenzen über die Verschmutzung der Umwelt bis zur direkten Gefährdung von Menschenleben. Wirtschaftliche Schäden werden beispielsweise durch die Zerstörung oder Beschädigung von Maschinen oder Produktionsgütern hervorgerufen, aber auch durch Produktionsausfälle, die aus einer Störung resultieren. Umweltverschmutzungen können durch das Austreten von schädlichen Stoffen auftreten. Insbesondere in der Chemie hat der Schutz von Mitarbeitern und Anwohnern einen großen Stellenwert, da es durch die Prozesse und Materialien zu schwerwiegenden Unfällen (Explosionen, giftige Gase, etc.) kommen kann [Web14]. Für die Integration neuer Ansätze und Lösungen muss nachgewiesen werden, dass diese ein mindestens gleichwertiges Sicherheitsniveau haben wie die Bestandslösung.

Lebenszyklen der Anlagen: Insbesondere in der Prozessindustrie liegen die Lebenszyklen von Anlagen im Bereich von mehreren Dekaden [FA09, VHDB13]. Eine Konsequenz dieser Tatsache ist, dass aktuell Anlagen mit den unterschiedlichsten Graden der Automatisierung und den verschiedensten Systemen betrieben werden. Dies macht die Wartung, Instandhaltung und Erweiterung zu schwierigen und herausfordernden Aufgaben. Insbesondere da nicht nur Hardwarekomponenten vorgehalten werden müssen, sondern auch die entsprechenden Engineeringtools zur Wartung der Automatisierungssysteme. Zusätzlich

müssen Mitarbeiter die dafür erforderliche Expertise besitzen [VHDB13, NAM08]. Eine aussagekräftige und aktuelle Dokumentation des Anlagenzustands ist bei diesen langen Lebenszyklen eine Herausforderung [Web14, WGE+17].

Unterbrechungsfreie Anlagenlaufzeiten: Ein weiteres Merkmal der Prozessautomation sind die langen unterbrechungsfreien Anlagenlaufzeiten. Zwischen zwei geplanten Abschaltungen können bei kontinuierlich produzierenden Anlagen bis zu drei Jahre liegen [FA09]. Da die Kosten durch einen ungeplanten Produktionsausfall hoch sein können, muss ein solcher Fall nach Möglichkeit vermieden werden.

Domänenspezifische Programmiersprachen: Die Programmiersprachen der IEC 61131 [IEC04] sind in der industriellen Automation sehr verbreitet und werden auf absehbare Zeit ihre Bedeutung behalten [WTE+17]. Die Sprachen sind in vielen Anwendungen über einen langen Zeitraum erprobt und die Mitarbeiter der Unternehmen verfügen über eine große Expertise in deren Nutzung.

Betrachtung von hybriden Soft- und Hardwaresystemen: Ziel der Produktionsautomation ist die Produktion von materiellen Gütern. Um dieses zu erreichen, sind Aktoren für die Produktion nötig. Diese repräsentieren die Schnittstelle zwischen der Cyberwelt und der physischen Welt. Ohne diese Hardwareanbindung ist eine Produktion materieller Güter nicht möglich. [HSF+13]. Um dies zu berücksichtigen, endet z. B. die Automatisierungspyramide (vgl. Kapitel 1) am unteren Ende mit der Anbindung ins Feld. Daraus resultiert, dass die aus der Hardware resultierenden Randbedingungen nicht zu vernachlässigen sind. Diese Kombination von Soft- und Hardwaresystemen führt zusammen mit den unterschiedlichen Anwendungsbereichen zu einer hohen Komplexität der Systeme [VHDF+14].

Vielfältige Automatisierungs- und Engineeringsysteme: Aufgrund des weiten Aufgabenspektrums und der angesprochenen langen Lebensdauer der Anlagen und ihrer Komponenten, werden unterschiedliche Arten von Automatisierungssystemen verschiedener Hersteller verwendet. Zur Entwicklung und Wartung der Automatisierungslösungen sind zusätzlich die kompatiblen Engineeringsysteme nötig. In Kombination mit dem nicht flächendeckenden Vorhandensein von Datenaustauschlösungen führt dies zu einem höheren Fehlerpotential, Ineffizienz und damit höheren Kosten bei der Anlagenplanung und damit der Planung von Automatisierungslösungen [VHDB13].

Heterogener Nutzerkreis der Systeme: Da die Automatisierungstechnik Schnittstellen zu verschiedenen Gewerken hat, werden die Systeme von Mitarbeitern mit unterschiedlichen Voraussetzungen (fachlicher Hintergrund, Ausbildungsniveau, Erfahrung, etc.) entwickelt, bedient und gewartet [VHDB13]. Entsprechend muss den jeweiligen Voraussetzungen und Rollen innerhalb der Organisationseinheit Rechnung getragen werden. Beispielsweise können spezialisierte Sichten auf das gleiche System vorgehalten werden [NAM14]. Trotz der fortschreitenden Automatisierung ist der Mensch weiterhin ein wichtiger Bestandteil im Produktionsprozess. Die Automatisierungstechnik entlastet den Bediener häufig, sodass dieser den Prozess insgesamt nach übergeordneten Gesichtspunkten wie Wirtschaftlichkeit und Sicherheit führen kann [Lun03].

3.2 Anforderungen an das Konzept

Abgeleitet aus den vorgestellten Eigenschaften der Domäne, werden im Folgenden die Anforderungen an das Konzept vorgestellt.

R1 Integration in bestehende Architekturen: Das Konzept zur Variantenbeschreibung muss sich in die viel verwendeten und erprobten Architekturen der Automatisierungstechnik einfügen, d. h., es muss sich nahtlos in die bestehenden Modelle der einzelnen Domänen einpassen. Wie in Abschnitt 2.2 beschrieben, sind komponentenbasierte Architekturen in der Automatisierungstechnik weit verbreitet. Diese finden zumindest implizit im Hard- und Softwarebereich Anwendung und bilden die Grundlage für die Entwicklung vieler Lösungen.

R2 Modellbasierte Abbildung der Variabilität Zur Förderung der Wiederverwendbarkeit in komponentenbasierten Architekturen muss die Variabilität der Lösungen durch einen modellbasierten Ansatz beschrieben werden. Modellbasierte Ansätze sind ein bewährtes Mittel zur Lösung von Aufgaben in der Automatisierungstechnik [WKS+16]. Insbesondere für den Austausch von Daten, aber auch für die Sicherstellung der Interoperabilität haben sich auf Metamodellen basierende Ansätze als geeignet erwiesen. Prominente Beispiele sind AutomationML [IEC16] und Merkmale [EMPA17].

R3 Brownfield Lösungen integrieren: Das Konzept muss angesichts der vielen bestehenden Lösungen und der langen Lebensdauer von Anlagen (vgl. 3.1) bestehende Lösungen integrieren. Darunter ist zu verstehen, dass das Konzept bestehende Lösungen als Ausgangspunkt für Wiederverwendung vorsehen soll. Es muss möglich sein, diese bestehenden (Teil-) Lösungen in neue Lösungen zu integrieren. Die bestehenden Systeme können nicht von einem Konzept ausgeschlossen werden, da sie aufgrund der langen Lebensdauer und der hohen Investitionskosten nicht durch Neusysteme ersetzbar sind. Ein Konzept, das keine bestehenden Lösungen integriert, wird sich schwerlich durchsetzen.

R4 Abstraktion von konkreten Implementierungen: Resultierend aus der heterogenen Tool-Landschaft muss das Konzept von konkreten Implementierungen abstrahieren. Das fördert zum einen den Austausch von Implementierungen, andererseits führt es zu einer Anwendbarkeit des Konzepts auf verschiedene Anwendungsfälle. Abstraktion bedeutet, von den herstellerspezifischen Systemen zu abstrahieren und eine gemeinsame Basis zu finden, auf der Wiederverwendung unabhängig von der konkreten Umsetzung des Systems möglich ist.

R5 Explizite Modellierung der Abhängigkeiten zwischen Varianten: Bestehende Abhängigkeiten zwischen Varianten müssen explizit modelliert werden. Der aktuelle Zustand, dass die Abhängigkeiten bestenfalls implizit in der Implementierung verborgen sind, reduziert sowohl die Produktivität als auch die Wiederverwendung. Ein Beispiel dafür ist das Kopieren eines Teilsystems, ohne dabei eine Referenz auf das Original zu erzeugen. Darüber hinaus ist es sinnvoll, derartig relevante Informationen allen Programmierern und Systemdesignern zugänglich zu machen bzw. aufgrund dieser Informationen Prozesse zum Propagieren von Änderungen zu starten.

R6 Explizite Darstellung der Beziehungen zwischen Varianten und Versionen: Analog zur Modellierung der Abhängigkeiten zwischen Varianten müssen auch die Beziehungen zu Versionen dargestellt werden. Die Modellierung, welche Version(en) einer Komponente zu einer Variante kompatibel sind, steht hier im Fokus. Die Umsetzung einer Versionierung ist in der Automatisierungstechnik ein relevantes Thema [VH09].

R7 Propagieren von Änderungen an (Software-)Varianten: In der Praxis werden an Softwaresystemen über ihren gesamten Lebenszyklus hinweg Änderungen vorgenommen. Dies können z. B. Fehlerbehebungen oder Erweiterungen des Funktionsumfangs sein. Im Kontext des Konzepts kann eine Änderung aus zwei Quellen resultieren: entweder ändert sich eine Komponente in einer Variante oder die Variante selbst ändert sich. In beiden Fällen muss ein Weg gefunden werden, diese Änderungen zu erfassen. Der Nutzer muss die Abhängigkeiten erfassen können und bei der Verarbeitung unterstützt werden.

R8 Integration in dezentrale Systeme: Eine Möglichkeit, die im Kontext von I4.0 geforderte Wandlungsfähigkeit zu realisieren, ist die Verwendung von modularen Anlagen bzw. Package Units. Diese können mit einer eigenen Steuerung ausgestattet sein und sich je nach Typ in ein überlagertes Leitsystem integrieren. Zusätzlich ist der vermehrte Einsatz von intelligenten Feldgeräten im Kontext von I4.0 im Gespräch. Die beiden genannten Entwicklungen führen zu einer vermehrten Verteilung der Steuerung auf unterschiedliche Geräte. Entwicklungen für modulare Systeme werden durch organisatorisch und räumlich getrennt arbeitende Personen durchgeführt. Die Entwicklung und Verwendung der Automatisierungslösungen erfolgten zunehmend dezentral. Diese dezentrale Entwicklung und Verwendung der Automatisierungslösung muss im Wiederverwendungskonzept berücksichtigt werden.

R9 Zentrale Variantenlagerung: Darüber hinaus ist es nötig, dass eine Austauschplattform für die Entwicklung mit mehreren Personen an einem System existiert. Diese Plattform ist auch im Hinblick auf den Erhalt von Wissen in einer Organisationseinheit und den Wissenstransfer wichtig. Sie muss zentral sein, um eine doppelte und inkonsistente Datenhaltung zu unterbinden.

R10 Integration in bestehende Prozesse: Das entwickelte Konzept muss sich in bestehende Prozesse zur Erstellung einer Lösung integrieren. Optimalerweise kann es additiv zu bestehenden Systemen verwendet bzw. an diese angeschlossen werden. Der hohe zeitliche Aufwand und zu hohe Kosten sind die Hauptgründe dafür, dass aktuell wenig variantenbasiert entwickelt wird [VHON18]. Für eine bessere Akzeptanz muss sich der Aufwand für die Integration in Grenzen halten.

R11 Verwendung bestehender Sprachen und Paradigmen: Da es in der Automatisierungstechnik anwendbar ist, muss ein Konzept für die Wiederverwendung die in Kapitel 2.1 vorgestellten Programmiersprachen berücksichtigen. Diese werden auf absehbare Zeit das Rückgrat der industriellen Automatisierungstechnik bilden.

R12 Automatisierte Interpretierbarkeit der zugrundeliegenden Modelle: Die Modelle des Konzepts müssen für eine (teil-)automatisierte Verwendung nutzbar sein. Dies ist für die Aufwandsreduktion bei der Erstellung von Automatisierungslösungen erforderlich. Diese Interpretierbarkeit ermöglicht die Verwendung von automatisierten Umwandlungen zur Übertragung der Modelle in verschiedene nutzerspezifische Visualisierungen. Dies ist im Hinblick auf den heterogenen Nutzerkreis in der Automatisierungstechnik nützlich.

R13 Anwendung des Konzepts auf hybride Systeme: Das Konzept muss für Hard- und Softwaresysteme gleichermaßen gelten, um dem Fokus der Automatisierungstechnik auf hybride Systeme gerecht zu werden. Wie in Abschnitt 2.1 dargestellt, werden in der Automatisierungstechnik Soft- und Hardware betrachtet. Insbesondere die Abhängigkeiten und Wirkzusammenhänge zwischen beiden Teilen eines Systems müssen berücksichtigt werden.

4 Stand der Wissenschaft

Am Anfang dieses Kapitels werden die Vorarbeiten des Autors für diese Arbeit zusammengefasst. Anschließend werden die Grundlagen der Wiederverwendung erläutert. Eine Definition von Variabilität und von möglichen Arten sie zu modellieren folgt im nächsten Unterkapitel. Anschließend wird die Delta-Modellierung aus der Softwaretechnik vorgestellt. Nachfolgend werden die Grundlagen der Modellierung eingeführt und erläutert. Dies beinhaltet beispielsweise die verschiedenen Ebenen der Modellierung, wie sie von der OMG festgelegt worden sind und die Modellebenen, die am Lehrstuhl für Prozessleittechnik verwendet werden. In diesem Zusammenhang werden Kriterien für gutes Modellieren aus der Literatur vorgestellt, die für diese Arbeit relevant sind. Abschließend werden unterschiedliche Sichten von Modellen, die bei der Modellierung eine Rolle spielen können vorgestellt.

4.1 Eigene Vorarbeiten

Anforderungen an die Automatisierung: In [WTE+17] werden die Anforderungen an ein Softwaresystem für die Automatisierung von großen prozesstechnischen Anlagen untersucht. Ausgehend von den Eigenschaften von Softwarequalität der IEC 25010 [ISO11] werden die Anforderungen ermittelt. In die Betrachtung werden die grundsätzlichen Anforderungen an die Automatisierungssysteme von großen prozesstechnischen Anlagen und die speziellen Anforderungen an die Automatisierung von Walzwerken eingeschlossen. Das Ergebnis ist ein Mapping von allgemeinen Softwarequalitätsmerkmalen auf die speziellen Anforderungen der Automatisierungstechnik. Die Liste ist mit einer internen Befragung von Entwicklern des Industriepartners abgeglichen worden. Die Arbeiten über die Anforderungen an Automatisierungslösungen [WTE+17] führten zu der Erkenntnis, dass die nicht-funktionalen Anforderungen von großer Bedeutung für die Akzeptanz der Tools sind. Diese Ergebnisse flossen in die Anforderungen an das in dieser Arbeit entwickelte Konzept ein.

Konzepte der Prozessführungsarchitektur: Ausgehend von den Arbeiten des Lehrstuhls für Prozessleittechnik in früheren Jahren (z. B. [Ens01, YQE10]) wird in [WE15a] das Konzept der „Sprechenden Kommandos“, d. h. von Menschen direkt verständlichen Prozessführungsaufträgen, eingeführt. Der vorgestellte Ansatz kann in die Ideen zur dienstebasierten Prozessführung im Kontext von Industrie 4.0 integriert werden. Im nächsten Schritt wird in [WE17] ein Vorschlag zu Integration von Diensten in IEC 61131 Architekturen vorgestellt. Ausgehend von Bausteinen für die Nutzung von Diensten aus der IEC 61499 wird das Interface eines Bausteins vorgestellt [WE17]. Zusätzlich wird ein Ablauf zur Erkundung der angebotenen Dienste und des Aufbaus des für den jeweiligen Dienst nötigen Interfaces entwickelt. Eine Herausforderung besteht darin, die Anlagen- und Prozessorientierung in einer

Architektur zu vereinen. Dafür wird in [WTPE17] ein Vorschlag für die Strukturierung von flexiblen Architekturen zur Prozessführung vorgestellt. Das Konzept wurde prototypisch realisiert und in verschiedenen Anwendungsszenarien erprobt. Wie die Konzepte an einem Beispiel angewendet und in der Lehre eingesetzt werden können, ist in [WSFE16] zu finden. Die Prozessführungsarchitektur bildet eines der Anwendungsgebiete für das Konzept zur Unterstützung der Wiederverwendung. Die Ergebnisse der Vorarbeiten in diesem Feld dienten insbesondere als Anwendungsfall für das Konzept. Zusätzlich konnten die Anforderungen an das Konzept durch die während der Realisierung der Prozessführungsarchitektur gewonnenen Erfahrungen präzisiert werden.

Wiederverwendung und Portabilität von Funktionsbausteinnetzwerken: In [WGE16] werden Mechanismen für die Wiederverwendung und Portabilität untersucht und ein Ansatz für die Verwendung in Anwendungen, die auf Funktionsbausteinnetzwerken basieren, vorgestellt. Hauptfaktoren für die Wiederverwendung und Portabilität sind in dieser Betrachtung die Ausführungssteuerung der einzelnen Komponenten eines Netzwerks und die Modellierung einer gemeinsamen Vorlage für die Anwendung. Mit der Beschreibung von Varianten in Bausteinnetzwerken wird in [WE15b] ein anderer Ansatz zur Wiederverwendung vorgestellt. Dabei wird ein Funktionsbaustein mit den Mitteln des Software Product Line Engineering beschrieben. Die vorgestellten Ansätze zur Wiederverwendung und Portierung wurden bei der Übertragung von Komponentensystemen aufgegriffen. Sie flossen in das Metamodell für die Beschreibung der Komponenten ein. Der Ansatz aus [WE15b] zur Beschreibung von Varianten bildete den Ausgangspunkt für die vorliegende Arbeit. Eine kritische Auseinandersetzung mit dem in [WE15b] Konzept führte zur Betrachtung der Delta-Modelle als Möglichkeit zur Beschreibung von Varianten.

Modellbasierte Entwicklung von Automatisierungssoftware: Neben dem flexiblen Aufbau einer Prozessführung (Automatisierungslösung) ist auch die Erzeugung eines solchen Aufbaus interessant. In [WKS+16] wird eine modellbasierte Herangehensweise für die automatisierte Generierung einer Automatisierungslösung vorgestellt. Dazu werden verschiedene Modelle (Merkmale, PandIX) verknüpft und analysiert, um zur Laufzeit eine Automatisierungsaufgabe lösen zu können. Welche Rolle der *Digitale Zwilling* oder die *Verwaltungsschale* im Engineering von Automatisierungssoftware spielen, wird in [WGE+17] untersucht. In diesem Beitrag wird die Verwendung und der Nutzen dieser neuen Konzepte anhand des Lebenszyklus einer Anlage nachvollzogen. Die Ansätze zur modellbasierten Entwicklung von Automatisierungslösungen bildeten die Grundlage für das in dieser Arbeit vorgestellte Konzept. Die Beschreibung der Komponentensysteme durch ein Metamodell und die Modellierung von deren Transformation durch Delta-Modelle ist eine Weiterführung der Ansätze aus [WKS+16].

4.2 Grundlagen der Wiederverwendung

Wiederverwendung ist ein universelles Konzept, das in den unterschiedlichsten Bereichen zum Einsatz kommt. Ziel ist es, Artefakte (Synonyme: Assets, Objekte) in verschiedenen Produkten oder Anwendungsfällen zu verwenden. Definition 2 ist [ISO11] entnommen und definiert das Qualitätsmerkmal Wiederverwendbarkeit von Systemen und Software.

Definition 2 (Wiederverwendbarkeit)**.** *Grad, mit dem ein Asset in mehr als einem System, Gebäude oder anderen Asset verwendet werden kann.*

Das Konzept Assets mehrfach und in verschiedenen Kontexten zu verwenden, wird sowohl in Soft- als auch Hardwaresystemen genutzt. Beispielsweise werden im Automobilbau baugleiche Teile in möglichst viele Fahrzeuge eingebaut. Vorteile dieses Ansatzes sind die geringeren Aufwände in der Entwicklung und Produktion der Fahrzeuge. So sind die Kosten geringer im Vergleich zur mehrfachen Entwicklung der Teile [Mey09]. Darüber hinaus entfallen die Tests nach einer Neuentwicklung und es muss nur ein Bauteil gepflegt werden anstatt mehrerer. In der Softwareentwicklung ist das Konzept ebenso bekannt. Dort entsteht ein großer Nutzen, da die Grenzkosten für Software marginal sind, d. h. jedes zusätzlich verwendete Asset (implementierte Instanz) ist nahezu ohne Kosten und Aufwände nutzbar.

Die Definition 3 für Wiederverwendung in der Softwaretechnik ist analog dazu (vgl. [Mey09]). Der Autor unterstreicht, dass die Wiederverwendung nicht nur ökonomische Vorteile bringt, sondern auch die Qualität der Software steigert. Die Vorteile der Wiederverwendung in der Softwaretechnik wird in [Lim94] untersucht und umfassen eine Verbesserung der Markteinführungszeit, der Produktqualität und der Produktivität.

Definition 3. *Wiederverwendbarkeit ist die Fähigkeit von Softwareelementen für die Konstruktion von unterschiedlichen Applikationen zu dienen.*

Die Herausforderungen bzw. die Hemmnisse für die Wiederverwendung bestehender Lösungen sind organisatorischer und menschlicher Natur [Mey88]. Zusätzlich ist die Entwicklung von wiederverwendbarem Code bis zu 480% teurer als von konventionellen Programmen [Lim94, Bör89].

Zusätzlich dazu wird von verschiedenen Autoren das *Not invented here Syndrom* als sehr großes Hemmnis bei der Wiederverwendung ausgemacht [Mey09, Bör89]. In [WES87] wird untersucht, ob Programmierer in der Lage sind, die Möglichkeiten der Wiederverwendung in Software korrekt einzuschätzen. Im Ergebnis schätzen die befragten Personen die Potentiale viel zu gering ein. Es wurde festgestellt, dass insbesondere ungeübte Programmierer die Potentiale nicht richtig einschätzen können. Eine Abhängigkeit vom Alter oder anderen Faktoren konnte nicht festgestellt werden.

In den folgenden Abschnitten wird eine kurze Einführung in das Thema Wiederverwendung gegeben. Es werden zunächst die Voraussetzungen für eine systematische Wiederverwendung und anschließend mögliche Arten der Wiederverwendung vorgestellt. Nach einem Überblick über Versionen und Versionierung schließt das Kapitel mit der Betrachtung der Wiederverwendung in der Automatisierungstechnik.

4.2.1 Gegenstand der systematischen Wiederverwendung

Die erste Frage, die bei Überlegungen zum Thema Wiederverwendung gestellt und beantwortet werden muss, ist: *Was ist der Gegenstand der angestrebten Wiederverwendung?* [KCH+92]. In [Die02] ist ein Überblick über verschiedene Objekte der Wiederver-

wendung in der Softwareentwicklung enthalten. Diese sind prinzipiell in allen Phasen der Softwareentwicklung zu finden. So ist die Wiederverwendung von Klassen-Spezifikationen und -Implementierungen, von Analyse- und Entwurfsmodellen sowie von Frameworks und Mustern im Kontext der objektorientierten Entwicklung möglich [GR95].

Gegenstände der Wiederverwendung können Programmfragmente, Muster, Prozeduren bzw. Funktionen, Module und Teilsysteme sein [Bör89]. Programmfragmente sind Stücke von Quellcode. Solche Gegenstände sind für die Wiederverwendung nur interessant, wenn sie z. B. wiederholt genutzt werden. Wiederverwendungsmechanismen auf Unikate anzuwenden ist nicht sinnvoll. Als Muster werden Vorlagen zur manuellen Erzeugung von Quellcode bezeichnet. Prozeduren und Funktionen sind Standard- oder Anwendungsprozeduren/-funktionen und Makros. Ein Modul im Sinne des Autoren ist ein Programmbaustein, der über eine definierte Schnittstelle mit der Außenwelt kommuniziert. Module zusammengefasst ergeben ein Teilsystem.

Der Nutzen der Wiederverwendung ist größer, je größer die Objekte der Wiederverwendung sind. Zusätzlich steigt der Nutzen je früher die Objekte im Lebens- bzw. Entwicklungszyklus eingesetzt werden [Bör89]. Der Autor betont allerdings, dass mit dem Ansteigen der beiden genannten Eigenschaften die Wahrscheinlichkeit sinkt, dass die Objekte die Anforderungen an eine spezifische Aufgabe erfüllen.

Die Unterscheidung zwischen der Wiederverwendung von Strukturen und der Wiederverwendung von Elementen wird in [Die02] vorgenommen. Für die Wiederverwendung von Strukturen nennt der Autor zwei Methoden: Die Adaptierung und die Spezialisierung. Adaptierung bezeichnet die Anpassung eines Produkts an ein konkretes Problem durch die Veränderung von Parametern. Im Gegensatz dazu wird bei der Spezialisierung eine vorhandene Funktionalität zur Erstellung eines neuen Artefakts genutzt.

Die Existenz einer inneren Architektur von Produkten stellt eine Voraussetzung für eine systematische Wiederverwendung dar [Sch16b]. In der Automatisierungstechnik besitzen die Systeme in der Mehrzahl eine innere Struktur (vgl. Kapitel 2.1.1). Zusätzlich werden in [Sch16b] mit der Baureihe, dem Baukasten, der Modulbauweise, der Plattform und der Produktfamilie Methoden für die Strukturierung von Produkten aus der Produktentwicklung vorgestellt. Eine Baureihe umfasst Produkte, die funktional ähnlich sind, sich aber in ihren Parametern, z. B. Leistungsdaten unterscheiden. Im Gegensatz dazu besteht ein Baukasten aus vorher entwickelten Komponenten, die zu unterschiedlichen Produkten zusammengefügt werden. Diese Produkte können sich beliebig stark unterscheiden. Die Modulbauweise ist eine Form des Baukastens, bei dem die Bausteine nicht mehr frei kombinierbar sind und einen erheblichen Teil der Gesamtfunktionalität ausmachen sollen. In der Plattform werden die drei Designmethoden zusammengefasst und kombiniert. Dabei wird insbesondere die Gemeinsamkeit zwischen verschiedenen Produkten in den Fokus gerückt. Diese Herangehensweise ist im Automobilbereich verbreitet. Die Produktfamilie betrachtet mehr als die Unterschiede zwischen den einzelnen Produkten, sie stellt die Gemeinsamkeiten der Produkte in den Vordergrund. Eine genauere Betrachtung von Produktfamilien und deren Variabilität ist in Kapitel 4.3 zu finden.

4.2.2 Arten der Wiederverwendung

Neben den Gegenständen der Wiederverwendung ist die Frage relevant, wie diese Gegenstände wiederverwendet werden. In der Literatur werden dafür verschiedene Mechanismen angeführt, die sich stark voneinander unterscheiden.

Die einfachste Art der Wiederverwendung ist das *Copy and Paste* [Sch16b]. Dabei wird ein bestehendes Artefakt kopiert und in einem anderen Anwendungsfall eingesetzt. Dieser Ansatz birgt jedoch das Risiko, dass der Kopiervorgang nicht adäquat dokumentiert wird und so mehrfach die gleiche Lösung entsteht, ohne dass diese Abhängigkeit vermerkt ist. Für die Wartung der Implementierungen ist dies nicht optimal, da so ein unnötiger Mehraufwand für die Pflege der einzelnen Instanzen entsteht. Durch eine fehlende Dokumentation der Abhängigkeiten werden die kopierten Instanzen möglicherweise nicht mehr gefunden und profitieren nicht von Verbesserungen. In [WGE16] wird die Kopiervorlage im Kontext von Funktionsbausteinanwendungen als Schablonen bezeichnet. Die einfache Implementierbarkeit und Anwendbarkeit dieses Mechanismus ist ein Grund für die Nutzung des Ansatzes.

Das beschriebene Defizit kann durch eine Verbindung zwischen der Schablone bzw. dem Prototypen und der Instanz behoben werden. Die Informatik kennt dafür zwei Arten der Typ-Instanz-Beziehung, nämlich die Prototyp-Instanz-Beziehung und die Klasse-Instanz-Beziehung [SDM95]. Die Prototyp-Instanz-Beziehung besteht zwischen der Schablone und den dazugehörigen Kopien. Vorteilhaft daran ist, dass Aktualisierungen zur Behebung von Fehlern an die Instanzen verteilt werden können. Darüber hinaus kann über die Verbindung nachvollzogen werden, wie die Instanzen mit den Prototypen zusammenhängen. Die zweite Beziehung ist die Klasse-Instanz-Beziehung. Hierbei handelt es sich um den Zusammenhang zwischen Klasse und den gebildeten Instanzen. Auf der Klassen-Ebene können Klassen voneinander abgeleitet werden. Die Frage, welches Paradigma besser (z. B. generischer anwendbar) ist, lässt sich nicht eindeutig beantworten. Eine Klasse-Instanz-Beziehungen ist zur Laufzeit unflexibel und stellt eine unnötige Einschränkung dar [SLU88]. Allerdings sind instanziierte Klassen im Allgemeinen an Ausführungsgeschwindigkeit überlegen [WGE16]. Typ-Instanz-Beziehungen kommen auch außerhalb der Softwaretechnik z. B. in der Konstruktionslehre zum Einsatz [Sch16b].

Für die abstrakte Wiederverwendung von Lösungsverfahren werden Pattern eingesetzt. Eine bekannte Sammlung von Design-Pattern aus der Softwaretechnik ist [GJHV11]. Die Pattern basieren auf Erfahrungswerten der Autoren und stellen in der Praxis gesammeltes Wissen der Autoren dar. Für jeweils eine Klasse von Problemen wird eine Lösung präsentiert, die vom Nutzer auf den jeweiligen Anwendungsfall angepasst werden muss. Die Nutzung von Strukturen für Regler (z. B. Kaskadenregler) ist ein Beispiel für die Anwendung von Pattern in der Automatisierungstechnik.

4.2.3 Versionen und Versionierung

Nach [Dud] ist eine *Version* eine „Ausführung, die in einigen Punkten vom ursprünglichen Typ oder Modell abweicht“. An dieser Definition werden bereits zwei wichtige Eigenschaften

von Versionen deutlich: Es gibt einen Ausgangsgegenstand, auf den sich die Version bezieht und zu diesem besteht eine Abweichung.

In [CW98] wird im Kontext der Konfigurationsverwaltung von Software eine Version v als $p = (p_s, v_s)$ definiert. p_s stellt einen Zustand in einer funktionalen Sicht dar und die Eingliederung in die Versionsfolge wird durch v_s repräsentiert. Diese Abfolge von Versionen wird als Versionsraum bezeichnet und besteht aus den einzelnen Versionen inklusive der Verbindungen zwischen diesen. In vielen Anwendungen wird der Versionsraum als Graph aus Knoten (Versionen) und Kanten (Verbindungen) dargestellt. Die Verbindungen zwischen den Versionen bilden Deltas, die die Unterschiede zwischen den Versionen beschreiben. Diese werden in gerichtet und ungerichtet unterteilt. Wenn es möglich ist, die Richtung der Veränderungen durch ein Delta zu invertieren, handelt es sich um ein ungerichtetes Delta. So ist es beispielsweise nicht möglich, das Löschen einer Codezeile rückgängig zu machen, wenn die Information, was in dieser Zeile stand, nicht erhalten worden ist.

Versionen von beispielsweise Softwareanwendungen werden aus verschiedenen Motivationen heraus entwickelt. Dazu gehört das Beheben von Fehlern, die Erweiterung der Funktionalität und das Einpflegen von Abhängigkeiten [CW98]. Die Autoren bezeichnen Versionen als Varianten, die dafür vorgesehen sind nebeneinander zu ko-existieren. Andere Versionen sind hingegen nicht für die parallele Verwendung vorgesehen. In der Automatisierungstechnik werden viele unterschiedliche Systeme verwendet und neue Versionen durch eine fehlende Beschreibung der Abhängigkeiten oft nur unvollständig ausgerollt. Dementsprechend ko-existieren in der Praxis Versionen, die dafür nicht vorgesehen sind.

Für die Entwicklung von Quellcode bieten sich dateibasierte Versionsverwaltungen insbesondere für das kollaborative Arbeiten sowie die Beschreibung und Dokumentation der Versionen an [Ott09]. Weitere Vorteile sind eine konsistente Datenhaltung und das Nachverfolgen von parallelen Entwicklungssträngen (Branches) [CSFP08]. In [Ott09] wird zwischen zentralen und dezentralen Architekturen von Versionsverwaltungssystemen unterschieden. Zentrale Architekturen bestehen aus einem zentralen Server, von dem die Versionen an die Clients verteilt werden und Änderungen an den Server zurückgesendet werden. Dezentrale Systeme verteilen die Versionsverwaltung lokal auf jeden Client. Ein Beispiel dafür ist *git*.

In Abbildung 4.1 ist ein Überblick über den SVN-Workflow zusammen mit den jeweils in den einzelnen Schritten verwendeten Kommandos dargestellt. Der Workflow wird als kontinuierlicher Integrationsprozess bezeichnet [ORA]. Erster Schritt ist das *check out* der Dateien aus der Versionsverwaltung. Wenn Änderungen am Quellcode durchgeführt worden sind, können diese in die Versionsverwaltung *commitet* werden. Um die lokale Version auf den Stand des Servers zu bringen, wird ein *update* durchgeführt. Durch *add*, *delete*, *copy* und *move* können die Dateien der Versionsverwaltung manipuliert werden. Durch das Zurückkehren zu einer Vorversion (*revert*) können Probleme gelöst werden, die durch den Wechsel auf die neue Version entstanden sind. Nach dem manuellen Lösen von Konflikten kann die Datei durch *resolve* als gelöst deklariert werden.

Die Behandlung von Versionen über die Grenzen der Gewerke in der Automatisierungstechnik ist eine der großen heutigen Herausforderungen [FFVH12, VH09]. In der Praxis stellt die Pflege und Migration von (Software-) Systemen eine große Aufgabe dar. Insbesondere die Abhängigkeiten zwischen den Versionen der einzelnen Komponenten und daraus resultierende Inkompatibilitäten erfordern viel Aufmerksamkeit.

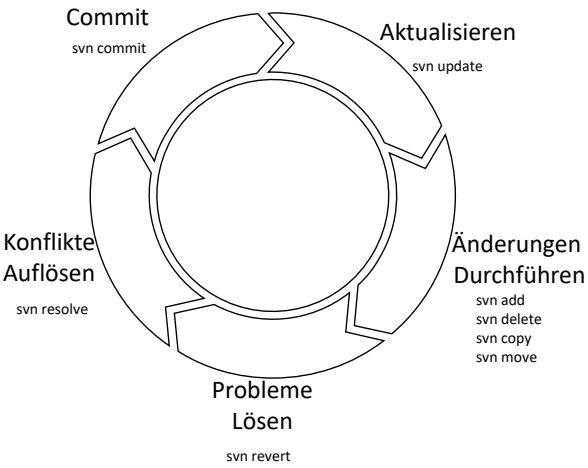


Abbildung 4.1: SVN Workflow nach [ORA]

4.2.4 Wiederverwendung in der Automatisierungstechnik

In Kapitel 2.1 ist der grundsätzliche Aufbau von industriellen Automatisierungssystemen beschrieben. Es ist zu erkennen, dass diese einen sehr weiten Bereich von Anwendungen abdecken. Er umfasst die Systeme zwischen dem Feld bzw. dem Prozess und der Betriebsleitebene. In diesem Bereich kommen viele Systeme, Konzepte und Architekturen zum Einsatz, die wiederum von unterschiedlichen Menschen entwickelt und betrieben werden. So arbeiten u. a. Informatiker, Ingenieure (verschiedener Fachrichtungen z. B. Elektrotechnik, Maschinenbau, Verfahrenstechnik, etc.), Chemiker und Physiker zusammen und bringen demzufolge die unterschiedlichen Herangehensweisen ihrer Fachdisziplinen in die Automatisierungssysteme ein [FFVH12]. Dies betrifft ebenso die verwendeten Ansätze zur Wiederverwendung in den unterschiedlichen Gewerken.

In den Softwaresystemen werden die typischen Konzepte der Wiederverwendung der Informatik angewendet. Eines dieser Konzepte ist das Klasse-Instanz Konzept aus der objektorientierten Programmierung, das aus dem Programmieren, Kompilieren und dem anschließenden Instanziieren sogenannter Klassen besteht. Grundsätzlich können einmal programmierte Klassen beliebig oft instanziiert werden. Funktionsbausteinarchitekturen sind eine Ausprägung dieser Form der Wiederverwendung [WGE16].

Wiederverwendung findet in der Automatisierungstechnik auf verschiedenen Ebenen und für verschiedene Zwecke statt. Dabei kann es sich z. B. um Desingpatterns und Konzepte, mit denen gute Erfahrungen gemacht wurden, handeln. Beispiele dafür sind Strukturen für den Einsatz von Reglern wie die Kaskade oder der Split-Range Regler. Ebenso werden realisierte (Teil-)Lösungen wiederverwende. Dabei kann es sich sowohl um Hardware als auch um Software handeln. Beispielsweise werden Hardwarekomponenten (z. T. standardisiert)

für verschiedene Anwendungen verwendet [DMG+17].

Die Wiederverwendung von Modulen von modularen Anlagen ist in [UDKO12] vorgestellt. Durch die automatisierte Einbindung von Modulen ist deren Wiederverwendung einfacher. In der Anlagenplanung existieren Ansätze zu einem wissensbasierten Engineering und einer Nutzung von Modularisierung [ODU13]. Grundlage der Wiederverwendung sind standardisierte Anlagenmodule.

Eine Wiederverwendung von Automatisierungsmodule ist in [Mah14] vorgestellt. Diese Module werden dezentral in einem Kommunikationssystem verteilt. Zusammen mit der verbauten Hardware werden sie zur Anwendung gebracht. Methodische Grundlage der Automatisierungsmodule ist die Verwendung von Pattern und Modulen, um so die Wiederverwendung zu unterstützen.

Zwei Ansätze zur Produktlinien-basierten Wiederverwendung (vgl. 4.3.1) für die Automatisierungstechnik werden in [Sch16b, FLK+14] vorgestellt. Beiden ist gemein, dass sie den Produktlinienansatz auf die Automatisierungstechnik anwenden. Die Autoren von [FLK+14] beschränkt sich bei der Betrachtung der Variabilität auf Feature-Modelle und damit den Problemraum [Sch16b]. Die Betrachtung des Lösungsraums erfolgt in [Sch16b]. Hier wird die Wiederverwendung durch eine Unterteilung des Konzepts in eine Entwicklung der Komponenten unabhängig vom Anwendungsfall und eine Anwendung auf das konkrete Problem unterstützt. Die Komponenten sind Gewerke-übergreifend definiert und als methodische Grundlage kommt das Product Line Engineering zum Einsatz.

4.3 Grundlagen der Variantenbeschreibung

Eine Möglichkeit für die Reduktion von Aufwänden und damit der Kosten im Engineering ist die Wiederverwendung bestehender Lösungen oder Teillösungen. Wiederverwendung im Kontext der Softwareentwicklung wird in [LS17] wie folgt definiert: *Prinzip der Objektorientierung, das zum Ziel hat, funktionsfähige Programmteile bereits bestehender Programme, sog. Module, in nachfolgenden Softwareprojekten wieder zu benutzen.* Vorteile der Wiederverwendung sind nach [LS17] eine Zeitersparnis in der Entwicklungszeit und die geringe Fehlerrate, da die Programmmodule bereits getestet und im Optimalfall in anderen Anwendungen erprobt sind. Ersetzt man in der genannten Definition Programm durch technische Komponente, Geräte oder Anlagenteil, so erhält man eine Definition von Wiederverwendung, deren Bedeutung über die Softwareentwicklung hinaus reicht. Der Nutzen ist in anderen Anwendungsfeldern vergleichbar zu der in der Softwaretechnik: es wird Zeit gespart und es kann auf erprobte Komponenten zurückgegriffen werden.

Um die Wiederverwendung von Lösungen zu unterstützen, haben sich je nach Anwendungsgebieten unterschiedliche Ansätze entwickelt, die den jeweiligen spezifischen Anforderungen und Randbedingungen Rechnung tragen. Im Folgenden werden mit dem Software Product Line Engineering und der Wiederverwendung in der diskreten Fertigung exemplarisch Wiederverwendungsansätze aus zwei ganz unterschiedlichen Domänen vorgestellt [CGR+12].

4.3.1 Varianten und Variabilität

Im folgenden Abschnitt werden Varianten eines materiellen und immateriellen Produktes vorgestellt. In der Herstellung von Industrie- oder Konsumgütern kommt es häufig vor, dass sich einzelne Produkte im Hinblick auf den Funktionsumfang oder den Aufbau nicht stark unterscheiden.

Begriffsdefinitionen

In [DIN02] ist die nachfolgende Definition von Variante zu finden. Die der Definition zugrunde liegende Norm wurde zurückgezogen, ist allerdings nach [BSG12] Grundlage für viele Überlegungen zu Varianten (vgl. [Lin94]).

Definition 4 (Varianten)**.** *Gegenstände ähnlicher Form und/oder Funktion mit einem in der Regel hohen Anteil identischer Gruppen oder Teile.*

Aus der Definition geht hervor, dass es sich bei Varianten um Gegenstände handelt, die eine ähnliche Form oder Funktion besitzen. Diese Gegenstände sollen aus identischen Teilen oder Gruppen bestehen. Nicht eindeutig ist jedoch, wie die Forderung nach „einem in der Regel hohen Anteil“ zu verstehen ist. Der Begriff der Variante ist in vielen der existierenden Definitionen unscharf und bedarf einer Präzisierung im konkreten Anwendungsfall. In [BSG12] kommen die Autoren zu dem Schluss, dass eine genaue Abgrenzung, welches Produkt Variante eines anderen ist, eine subjektive Festlegung ist.

Zur Entscheidung, ob ein Gegenstand ein eigenständiges Produkt oder Variante eines anderen Produkts ist, ist die Festlegung von Vergleichskriterien und eines maximalen Abstands zwischen Produkten nötig [BSG12]. Die Vergleichskriterien sind Merkmale, die die betrachteten Gegenstände charakterisieren. Unter Einbeziehung dieser Vergleichskriterien muss festgelegt werden, wie ähnlich sich zwei Gegenstände sein müssen, um als Varianten eines Produktes zu gelten.

In [Lin94] wird eine Unterteilung in technische und strukturelle Varianten vorgenommen. Technische Varianten variieren hinsichtlich der Geometrie, des Materials oder der Technologie eines Gegenstands. Bei aus mehreren Teilen bestehenden Gegenständen können strukturelle Varianten durch Einbindung von verschieden Komponenten in ein Produkt entstehen. Diese Charakterisierung wird in [DIN02] noch um die Funktion des Gegenstandes erweitert. Im Ergebnis umfassen die Vergleichskriterien die Funktionalität, den Aufbau und die verwendeten Technologien. In [BSG12] wird festgestellt, dass es keine abschließende Liste von Merkmalen zu einem Gegenstand geben kann. Vielmehr ist es in jedem Einzelfall erforderlich, die Ähnlichkeit anhand von relevanten Merkmalen, die eine Unterscheidung der betrachteten Produkte zulassen, zu prüfen.

Wie groß der Abstand zwischen den Gegenständen maximal sein darf, damit diese als Varianten gelten, wird in der Literatur diskutiert. Nach [Lin94] entsteht die neue Variante eines bestehenden Produkts durch Veränderung von einem oder mehreren Merkmalen. Jedoch wird eingeschränkt, dass in der praktischen Anwendung nicht immer eindeutig ist, ob ein Gegenstand eine Variante oder ein neues Produkt ist. In die gleiche Richtung wird in [Brä04] argumentiert. Varianten werden ebenso über die Gleichheit von Eigenschaften

definiert. Auch bei der Untersuchung der Ähnlichkeit kommt [BSG12] zu dem Ergebnis, dass Gleichheit von Merkmalen von Gegenständen und demzufolge die Betrachtung von Gegenständen als Varianten eines Produkts subjektiv ist.

In [PBL05] wird eine Variante als Repräsentation des variierten Gegenstandes verstanden. Die Autoren empfehlen einen dreistufigen Prozess bei der Festlegung von Varianten. Zunächst muss der Gegenstand gewählt werden, der variiert werden soll. Im zweiten Schritt werden das Merkmal oder die Merkmale identifiziert, die sich ändern sollen. Im letzten Schritt werden die eigentlichen Varianten festgelegt, die durch eine Änderung des Merkmals erreicht werden sollen.

Ausgehend von den vorgestellten Betrachtungen wird in der vorliegenden Arbeit ein Gegenstand als **Variante** eines Produkts verstanden, wenn er diesem in der Mehrzahl der für die Betrachtung relevanten Merkmalausprägungen gleicht. Jeweils zwei Varianten müssen sich in mindestens einer Merkmalausprägung unterscheiden. Die relevanten Merkmale können technischer, funktionaler und struktureller Art sein.

Die verwendete Definition von einer Variante ist analog zu der in [Sch16b] vorgestellten. Allerdings ist die Bindung des Variantenbegriffs an die Produktlinie für die vorliegende Arbeit nicht zweckmäßig. Die in dieser Arbeit verwendete Definition kann als Verallgemeinerung derjenigen von Schröck betrachtet werden.

Ausgehend von den beschriebenen Vergleichskriterien werden Varianten in unterschiedliche Arten unterteilt. Wie beschrieben werden die Varianten durch [Lin94] in technische und strukturelle Varianten unterteilt. Denkbar sind unterschiedliche Blickwinkel für Varianten: Arbeitsfluss, Architektur und Verhalten [KLL+14]. Alternativ dazu werden Strukturvarianten, Teilevarianten, Mengenvarianten und Funktionsvarianten unterschieden [Sch16b]. Die vorgestellten Arten von Varianten sind nicht orthogonal zu einander, d. h., sie sind nicht disjunkt. Beispielsweise können sich zwei Funktionsvarianten zusätzlich zu den Unterschieden in der Funktionalität strukturell unterscheiden. Die Einteilung kann dementsprechend nur den Hauptbetrachtungsgegenstand wiedergeben und ist ebenfalls stark subjektiv. Weitere Arten der Varianteneinteilung sind in [Brä04] zu finden.

Eng verknüpft mit dem Begriff der Variante ist die **Variabilität**. Nach [PBL05] bedeutet Variabilität umgangssprachlich die Fähigkeit oder die Tendenz zur Änderung. Ausgehend von dieser Betrachtung kommen die Autoren zu drei elementaren Fragen, die mit der Variabilität assoziiert sind:

- Was variiert? (Variabilitätssubjekt)
- Warum variiert es?
- Wie variiert es? (Variabilitätsobjekt)

Die erste Frage bezieht sich auf den Gegenstand der Variation, d. h. der sich ändernde Teil oder das sich ändernde Merkmal eines Gegenstandes in der wirklichen Welt. Dies wird als Variabilitätssubjekt bezeichnet. Die zweite Frage bezieht sich auf den Grund der Änderung. Mögliche Gründe sind sich ändernde Anforderungen der Stakeholder, unterschiedliche gesetzliche Rahmenbedingungen oder technische Gründe. Die letzte Frage nach dem „Wie?“ zielt auf die Menge an Ausprägungen, die ein Variabilitätssubjekt annehmen kann. Jede Ausprägung wird als Variabilitätsobjekt bezeichnet.

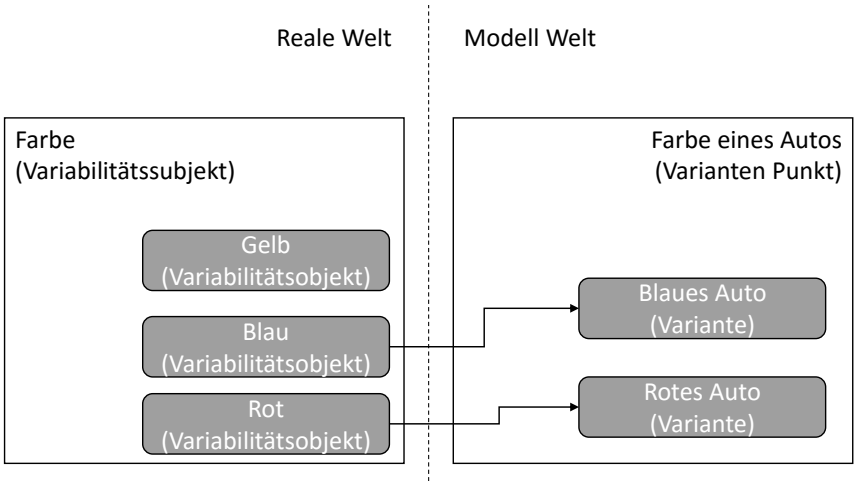


Abbildung 4.2: Übersicht über die Variabilität anhand des Beispiels der Autofarbe nach [PBL05]

In Abbildung 4.2 sind die vorgestellten Begrifflichkeiten am Beispiel der Autofarbe dargestellt. Die Darstellung unterscheidet reale Welt und Modell. Die Farbe des Autos in der realen Welt ist das Variabilitätssubjekt und im Modell der Variantenpunkt. Die möglichen Ausprägungen der Farbe in der realen Welt (Gelb, Blau, Rot) sind die Variabilitätsobjekte. Analog dazu werden das blaue Auto oder das rote Auto im Modell als Varianten bezeichnet.

Ebenso wie in [Sch16b] wird in der vorliegenden Arbeit der Begriff Variante nicht nur für das Modell eines Produkts, sondern auch für das reale Objekt verwendet. Zum einen kann die Unterscheidung zwischen der Variante im Modell und der realen Variante zu Verwirrung führen, zum anderen gibt es keinen verbreiteten Begriff für die reale Variante.

Die Beherrschung der Variantenvielfalt und die Unterstützung des zielgerichteten Entwicklungsprozesses wird als Variantenmanagement bezeichnet [DB+07]. Ziel ist es, Produkte, die sich nur wenig unterscheiden (Produktvarianten), ökonomisch sinnvoll herstellen zu können [Ava06]. Ein Bestandteil des Managements von Variantenvielfalt ist die Modellierung der auftretenden Varianten bzw. der Variabilität in Varianten-Modellen. In diesen werden „die Gemeinsamkeiten und Variabilitäten der Artefakte eines Systems mit dem Organisations- und Domänen-spezifischen Merkmalen und Abhängigkeiten“ beschrieben (vgl. [SRC+12]). Im Folgenden Abschnitt werden verschiedene Arten der Variabilität vorgestellt.

Im Kontext von Software Produktlinien Engineering wird in [VG07] das **Variantenmanagement** als Vorgehen zur Identifizierung, zum Design, zur Implementierung und zur Verfolgung von Flexibilität in Software Produkt Linien definiert. Im Rahmen der vorliegenden Arbeit wird die Bedeutung von Variantenmanagement nicht nur

im Zusammenhang mit Softwaresystemen gesehen, sondern zusätzlich auf hybride und Hardwaresysteme ausgedehnt.

Arten der Variabilität

Angelehnt an die Frage „Warum variiert das Variabilitätssubjekt?“ kann Variabilität anhand ihrer Ursache in die interne und die externe Variabilität unterteilt werden. Diese Unterscheidung dient nicht nur der Klassifikation der Variabilität, sondern soll darüber hinaus auch ein Bewusstsein dafür schaffen, für welchen Beteiligten die konkrete Variabilität relevant ist. **Externe Variabilität** ist für den Kunden sichtbar, z. B. die Farbe eines Autos (vgl. [PBL05]). Im Gegensatz dazu ist die Verwendung zweier Arten von funktional identischen Bremsbelägen für den Kunden nicht direkt sichtbar. Diese Variabilität wird als **interne Variabilität** bezeichnet. Man könnte also folgern, dass mit externer Variabilität das Ziel verfolgt wird, einen höheren Kundennutzen durch die Produktion von auf den Kunden direkt zugeschnittenen Produkten zu erzielen. Weitere Gründe für externe Variabilität können landesspezifische gesetzliche Regelungen oder Standards sein [PBL05]. Mit interner Variabilität können beispielsweise Bestandteile von hochpreisigen und günstigeren Produkten verwendet werden, ohne vom Kunden einsehbar zu sein. Ebenso kann es sein, dass der Hersteller die Komplexität für den Kunden reduzieren möchte und daher nur höherwertige Wahlmöglichkeiten bereitstellt. Die Details der technischen Umsetzung und mögliche weiterführende technische Auswirkungen werden dabei vor dem Kunden verborgen. Zusätzlich zu den bereits beschriebenen Faktoren sind die Business- und die Marketingstrategie des Herstellers entscheidend für die Einteilung in interne und externe Variabilität [PBL05].

Im Bereich der industriellen Automation ist die Unterscheidung von interner und externer Variabilität nicht einfach, da die Abgrenzung zwischen dem Hersteller eines Produkts und dem Kunden nicht immer eindeutig ist. Die Automatisierungslösungen werden möglicherweise nicht durch den Betreiber selbst erstellt, allerdings fordert dieser in der Regel Zugriff auf die Implementierung und wartet oder erweitert diese selbst. Darüber hinaus ist zur Optimierung oder Wartung von Anlagen eine hohe Expertise der Anlage erforderlich, sodass es nicht sinnvoll ist, vor den Nutzern etwas zu verbergen [Sch16b]. Zusätzlich ist die Verwendbarkeit für den Endnutzer auch nicht immer im Fokus der Produktentwicklung (Hard- und Software) [WTE+17]. Allerdings ist es für die Reduktion der Komplexität für den Nutzer an manchen Stellen sinnvoll, eine Unterteilung in interne und externe Variabilität vorzunehmen.

Zwei weitere Dimensionen der Einteilung von Variabilität sind die „Variability in Space“ und die „Variability in Time“ (vgl. [PBL05]). Bei der Variability in Space handelt es sich um verschiedene Ausprägungen einer Funktionalität, die je nach Präferenz des Nutzers oder den gegebenen Randbedingungen eingesetzt werden. Ein Beispiel dafür ist der Authentifizierungsmechanismus für Smartphones. Inzwischen verfügt der Nutzer je nach Modell über die Möglichkeit eine Pin einzugeben, sich mit seinem Fingerabdruck auszuweisen oder sich bei neuen Geräten mit Hilfe der eingebauten Kamera über das Aussehen zu identifizieren. Unter Variability in Time ist eine Variabilität über einen gewissen Zeitraum zu verstehen, z. B., weil eine Technologie durch eine andere abgelöst wird. Die Einführung von digitalen Kameras ist ein Beispiel dafür.

4.3.2 Variabilitätsmodelle

Um die Variabilität in der Praxis nutzen zu können, ist es wichtig zu verstehen, auf welche Arten Variabilität modelliert werden kann. Jedes Modell der Variabilität hat sein eigenes Anwendungsgebiet und kann dort einen Mehrwert schaffen. Variabilitätsmodelle können in verschiedene Arten unterteilt werden. Bekannt ist die Unterteilung in Problemraum und Lösungsraum [SRC+12].

Problemraum: Variabilität im Problemraum bezeichnet die domänenspezifische Variabilität, d. h. die Variabilität, die beispielsweise der Kunde eines Produktes wahrnimmt. Diese Form der Variabilität beschreibt eine funktionale Sicht auf ein Produkt. Die Modellierung der Variabilität erfolgt vorwiegend Feature- oder Entscheidungs-orientiert. Die Variabilität im Problemraum wird als Produktlinien-Variabilität bezeichnet [MPH+07].

Lösungsraum: Variabilität im Lösungsraum, auch Software Variabilität genannt, ist die Variabilität der wiederverwendbaren Artefakte [MPH+07]. Die Variabilität im Lösungsraum betrachtet die Variabilität bei der Erstellung der Lösung. Beispiele für solche Artefakte sind Architekturelemente, Testfälle, Komponenten und Dokumente [SRC+12]. Diese Art der Variabilität ist aus der Entwicklung von einzelnen Softwaresystemen bekannt [MPH+07]. So stellt die Spezialisierung einer Klasse eine Form der Variabilität im Lösungsraum dar. Bei der Spezialisierung werden Teile der Superklasse in einem weiteren Produkt (der abgeleiteten Klasse) verwendet.

Die interne und externe Variabilität unterscheidet, ob eine Variation vom Kunden wahrgenommen wird oder nicht. Im Gegensatz dazu wird mit der Differenzierung zwischen Problem- und Lösungsraum zwischen der Funktionalen- und der Implementierungssicht unterschieden. Die beiden Klassifizierungsansätze sind nicht disjunkt, stellen aber unterschiedliche Unterscheidungsmerkmale in den Vordergrund.

Für die Modellierung der Variabilität sowohl im Problem- als auch im Lösungsraum sind unterschiedliche Eigenschaften relevant. Dementsprechend gibt es verschiedene Ansätze die Variabilität in Modellen zu beschreiben und diese formalisiert festzuhalten. Im Folgenden werden zunächst Variabilitätsmodelle für den Problemraum vorgestellt. Anschließend wird auf die Variabilitätsmodelle des Lösungsraums eingegangen.

Variabilitätsmodelle für den Problemraum

Mit den Feature- und Entscheidungs-orientierten Modellen existieren zwei Klassen der Variabilitätsmodellierung im Problemraum [SRC+12]. Diese zwei Modellarten werden im Folgenden vorgestellt.

Das *Feature-Modell* beschreibt eine Klasse von Produkten (Produktlinie) als hierarchische Kombination von möglichen Merkmalen bzw. Eigenschaften. Die Merkmale werden in dem Modell baumartig angeordnet. Die Abhängigkeiten zwischen den Eltern-Merkmalen und Kinder-Merkmalen können nach [Bat05] in folgende Klassen unterteilt werden: *Und*, *Alternativ*, *Oder*, *Verpflichtend* und *Optional*. Das Feature-Diagramm wird auch als graphische Repräsentation des Merkmalbaums bezeichnet [KCH+90]. Zusätzlich werden die Abhängigkeiten zwischen den Merkmalen, sowie die Dokumentation der Designentscheidungen

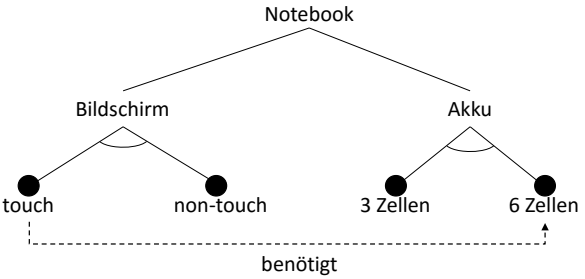


Abbildung 4.3: Beispiel für ein Feature-Diagramm.

und aller existierenden Merkmale, als Bestandteile des Modells gesehen. Abhängigkeiten zwischen Merkmalen können die notwendige Verwendung oder der Ausschluss eines weiteren Merkmals sein. In Abbildung 4.3 ist beispielhaft ein Feature-Diagramm dargestellt. Es ist zu erkennen, dass die Produktlinie *Notebook* aus den Merkmalen *Bildschirm* und *Akku* besteht. Als Bildschirm kann entweder ein *touch* oder ein *non-touch* Modell verbaut werden. Für die Energieversorgung stehen ein drei- oder ein sechs-Zellen Akku zur Verfügung. Die Wahlmöglichkeiten sind jeweils Alternativen, was durch den nicht ausgefüllten Kreisausschnitt zwischen den Linien verdeutlicht wird. Die ausgefüllten Kreise zeigen an, dass die Merkmale verpflichtend sind, z. B., dass ein Bildschirm in einem Notebook sein muss. Durch die gestrichelte Linie wird ausgedrückt, dass bei der Verwendung eines touch-Bildschirms ein großer Akku verwendet werden muss.

Im Rahmen eines *entscheidungsorientierten Ansatzes* wird versucht, für jede Variante eine Frage zu stellen, deren Beantwortung zur Auswahl der Varianten führt [SRC+12]. Dies können beispielsweise einfache Fragen sein, die mit ja oder nein zu beantworten sind. Im Fall von komplexeren Entscheidungen, in denen mehr als eine Option anwendbar ist bzw. mehr als eine Variante zeitgleich verwendet werden kann, bietet sich die Verwendung von solchen Entscheidungsfragen an. Analog zu den Feature-Modellen werden die Randbedingungen ebenfalls modelliert.

Es ist zu erkennen, dass beide Modelle eine Produktlinie aus einer funktionalen Sicht betrachten. Während das Feature-Modell alle Merkmale deskriptiv festhält, versucht der entscheidungsorientierte Ansatz den Nutzer durch geeignete Fragen zu dem für ihn passenden Produkt einer Produktlinie zu führen. Beiden Modellen ist gemein, dass sie von der konkreten Umsetzung der Produkte abstrahieren. Ein Überblick und ein Vergleich der beiden Modelle sind in [CGR+12] zu finden. Im Folgenden werden Variabilitätsmodelle zur Beschreibung des Lösungsraums betrachtet.

Variabilitätsmodelle für den Lösungsraum

Die Modelle der Variabilität im Lösungsraum werden anhand des Vorgehens beim Lösungsaufbau klassifiziert [Sch16b]. In [VG07] werden der annotative und der kompositionelle An-

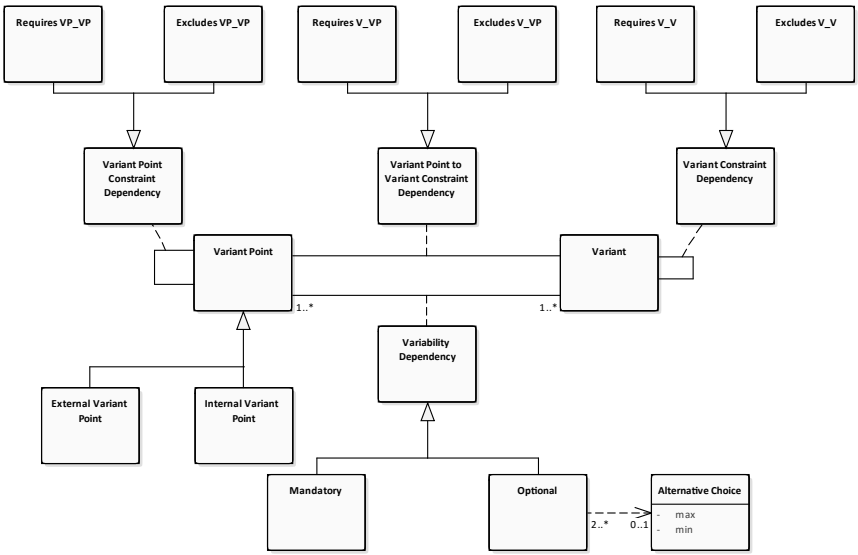


Abbildung 4.4: UML-Modell der Variabilität nach [PBL05]

satz unterschieden. *Annotative Ansätze* werden auch als *150% Lösungen* bezeichnet und bestehen aus einem Modell, in dem die gesamte Variabilität einer Produktlinie dargestellt ist. Um ein gültiges Produkt aus der Produktlinie zur erzeugen, werden die nicht benötigten Artefakte aus dem Modell entfernt. Zur Beschreibung dieser Modelle werden meist formale oder semi-formale Ansätze verwendet [Sch16b]. *Kompositionellen Ansätzen* liegt das entgegengesetzte Vorgehen zugrunde. Bei ihrer Verwendung wird das konkrete Produkt aus Artefakten zusammengestellt. Dies wird dadurch erreicht, dass die Additive um einen Produktkern, der allen Produkten einer Produktlinie gemein ist, angeordnet werden [SRC+12]. In [Sch16b] wird darauf hingewiesen, dass sichergestellt sein muss, dass die einzelnen Artefakte miteinander kombinierbar sind. Andernfalls besteht die Gefahr von hohen Aufwänden für Nacharbeiten an den Schnittstellen.

Eine Kombination aus annotativen und kompositionellen Ansätzen ist der **transformative Ansatz**. Er vereint die Reduktion und Addition und erlaubt so, Produkte einer Produktlinie entsprechend aufgestellter Regeln zu transformieren. Die Delta-Modellierung ist ein Beispiel für einen transformativen Ansatz [SRC+12]. Sie definiert die Operationen *Addition*, *Subtraktion* und *Modifikation*, um Produkte einer Produktlinie ineinander umzuwandeln. Sie ist ein intuitiv verständlicher Ansatz, der das Potential besitzt, in der Automatisierungstechnik gut einsetzbar zu sein. Aufgrund des transformativen Charakters kann die Delta-Modellierung nahtlos in bestehenden Systemen eingesetzt werden. Ein detaillierter Überblick über die verschiedenen Variabilitätsmodelle ist in [SRC+12] zu finden. In Abschnitt 4.3.3 wird die Delta-Modellierung und ihre Anwendungen in der Softwaretechnik vorgestellt

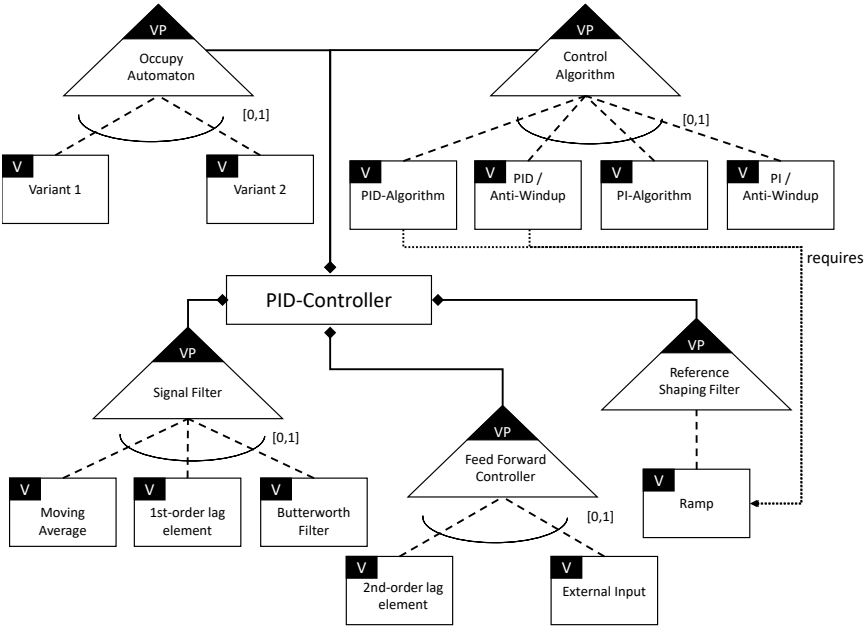


Abbildung 4.5: Beispiel für die Nutzung eines Orthogonalen Variabilitätsmodells für die Beschreibung eines PID-Regler Funktionsbausteins [WE15b]

Eine weitere Möglichkeit Variabilität zu modellieren, die die Problem- und Lösungsraumraumdarstellung kombiniert, ist das *orthogonale Variabilitätsmodell*. In einem orthogonalen Variabilitätsmodell wird das Produkt, bzw. die Produktlinie, aus funktionaler Sicht beschrieben. Es werden alle Variantenpunkte der Produktlinie mit den dazugehörigen Varianten aufgeführt. In Abbildung 4.4 ist das UML-Modell des Zusammenhangs zwischen Variantenpunkten und Varianten nach [PBL05] dargestellt. Darin ist zu erkennen, dass ein Variantenpunkt sowohl intern als auch extern sein kann. Die Zuordnung einer Variante zu einem Variantenpunkt ist entweder verpflichtend oder optional. Wenn die Zuordnung optional ist, kann eine Wahl aus mehreren Alternativen erzwungen werden. Dies wird über die Parametrierung der alternativen Wahl (min und max) realisiert. Zusätzlich ist im Modell die Abbildung von Abhängigkeiten zwischen Variantenpunkten, zwischen Varianten und zwischen Variantenpunkten und Varianten vorgesehen. Die Abhängigkeit kann entweder das Benötigen oder das Ausschließen des jeweils anderen Elements sein. Dies ist als Randbedingung für den Aufbau von erlaubten Produkten zu verstehen. Wenn das Ausgangselement in einem Produkt enthalten ist, wird das Zielelement entweder benötigt oder dessen Verwendung ausgeschlossen.

In Abbildung 4.5 ist die Nutzung des vorgestellten UML-Modells am Beispiel eines PID-Regler Funktionsbausteins dargestellt. Es ist zu erkennen, dass der Funktionsbaustein durch fünf Variantenpunkte modelliert ist. Namentlich sind es der Belegungsautomat, der

Regelalgorithmus, der Signalfilter, der Anschluss an den Feedforward Controller und der Sollwertfilter. Die Variantenpunkte werden durch eine bis vier Varianten realisiert. Wenn mehr als eine zugeordnete Variante existiert, sind die Varianten als alternative Wahlmöglichkeit dargestellt. Es kann jedoch maximal eine Variante ausgewählt werden. Bei der Wahl des PID-Algorithmus wird die Nutzung eines Rampenfilters erzwungen.

Das orthogonale Variabilitätsmodell findet im *Software Produktlinien Engineering* Anwendung. Das Software Produktlinien Engineering ist ein Prinzip zur Entwicklung und Implementierung von Softwarelösungen. Ziel ist es, ähnliche Produkte aus einer Menge von gleichen Komponenten zu bauen bzw. zu implementieren. Die Ideen dazu stammen zum Teil aus der Großserienfertigung der Automobilindustrie. Die Zielsetzung ist es, Produkte um einen Kern herum zu entwickeln und so Softwarekomponenten mehrfach zu verwenden. Zur Erreichung dieses Ziels wurden Entwicklungsprozesse und Vorgehensweisen entwickelt, die die Wiederverwendung unterstützen und fördern sollen sowie die Varianten in den Mittelpunkt des Entwurfes rücken. Der Fokus liegt dabei auf der Berücksichtigung der Gemeinsamkeiten von verschiedenen Produkten insbesondere im Hinblick auf deren Funktionalität [KLD02].

Zusammenhang zwischen Modellen des Problem- und Lösungsraums

Die Variabilitätsmodelle des Problem- und Lösungsraums beschreiben unterschiedliche Gesichtspunkte von Produktlinien. Schlussendlich stellen sie aber nur zwei unterschiedliche Betrachtungsweisen des gleichen Gegenstandes dar. Um die Modelle sinnvoll nutzen zu können, muss die Beziehung zwischen den Modellen der beiden Räume beschrieben sein [SRC+12]. Durch diese Verbindung kann der Kunde bzw. der Nutzer eines Produkts aus der Produktlinie die von ihm benötigten Funktionalitäten auswählen und es kann das entsprechende Produkt zusammengefügt werden. Dafür muss modelliert werden, welche Artefakte im Lösungsraum die jeweiligen Funktionalitäten im Problemraum realisieren. In [CHS10] wird der Zusammenhang zwischen Problemraum und Lösungsraum als *Anwendungsfunktion* bezeichnet. Ein Überblick über verschiedene Ansätze zur Modellierung der Verbindung ist in [SRC+12] zu finden.

4.3.3 Delta-Modelle in der Softwaretechnik

Grundsätzlich sind Delta-Modelle ein transformativer Ansatz, durch den der Unterschied zwischen mindestens zwei Gegenständen beschrieben werden kann. Eine sehr anschauliche Anwendung von Deltas ist das Bilden der Differenz zwischen zwei Programmständen. Im Ergebnis erhält man zeilenweise die Information, ob eine Zeile hinzugefügt oder gelöscht wurde. Mit diesem Vorgehen können Änderungen am Quellcode sehr leicht erfasst und nachvollzogen werden. Tools zur Unterstützung dieses Vorgehens sind heutzutage Standard bei den gebräuchlichen Codeverwaltungsanwendungen (z. B. GitHub, SVN, etc.).

Im Kontext der Modellierung von Variabilität ist die Deltamodellierung eine Ausprägung der transformativen Modelle des Lösungsraums. Die erste Definition der Delta-Modelle für Softwareproduktlinien ist in [Sch10] zu finden. Ausgehend von der UML-Komponente (vgl. Kapitel 2.2) beschreibt die Autorin ein formalisiertes Delta-Modell. Das Modell besteht aus

Elementen und Verbindungen zwischen den Elementen. Aus den Elementen und Verbindungen können Systeme gebildet werden, die definierte Merkmale erfüllen. Ist ein System eine valide Konfiguration aus Merkmalen, so wird es nach [Sch10] je nach Verwendung als *Kernmodell* bzw. *Produktmodell* bezeichnet.

Ausgehend von einem Ausgangssystem definiert ein Delta-Modell eine Menge von Operationen. Diese Operationen beschreiben die Modifikationen an dem Ausgangssystem und überführen es in das Zielsystem. Zusätzlich beschreibt das Delta-Modell eine sogenannte Anwendungsbedingung. Dies ist eine Vorbedingung, die im Hinblick auf die Merkmale des Ausgangsmodells vorliegen muss, damit das Delta-Modell anwendbar ist. In [Sch10] werden fünf Operationen für die Modifikation von Ausgangsmodellen definiert:

- **add Element**
  Die Operation add fügt ein Element zu dem Ausgangsmodell, auf das sie angewendet wird, hinzu.
- **mod Element**
  Durch die Operation mod wird ein Element modifiziert. Dies könnte beispielsweise die Änderung eines Parameters oder des internen Zustands des Elements sein.
- **rem Element**
  Das Entfernen eines Elements wird durch die Operation rem realisiert.
- **add Verbindung**($Element_1$,$Element_2$)
  Durch diese Operation wird eine Verbindung zwischen zwei Elementen angelegt.
- **rem Verbindung**($Element_1$,$Element_2$)
  Die Operation rem entfernt eine Verbindung zwischen zwei Elementen aus dem Modell.

In Abbildung 4.6 ist ein Überblick über die verschiedenen Artefakte und Begriffe sowie ihre Beziehungen dargestellt. Ausgangspunkt ist die Produktlinie, die aus einem Featuremodell (Φ), einem Kernprodukt (c), einem Delta-Modell ($D, \prec$) und einer Applikationsfunktion (γ) besteht. Aus dieser Produktlinie ist ein Delta-Modell entnommen, das eine benötigte Menge an Merkmalen erfüllt. Der Weg zur Erzeugung eines Produkts selbst kann aus mehr als einem einfachen, z. B. einem zusammengesetzten Delta-Modell bestehen. Dies könnte eine Aneinanderreihung von Delta-Modellen sein, die von einem Kernprodukt bis zu einem Produkt führen. Für die Anwendung des Deltas bzw. der in ihm enthaltenen Delta-Operationen gibt es grundsätzlich zwei Möglichkeiten: Die Anwendung auf ein Kernprodukt oder der Aufbau eines Produkts aus Delta-Operationen, ohne dass ein Kernprodukt als Ausgangspunkt existiert. Werden die Delta-Operationen auf ein Kernprodukt angewendet, wird dadurch ein neues Produkt erzeugt. Die Variabilität aus dem Problemraum (Featuremodell) wird so im Lösungsraum durch ein neues Produkt abgebildet. Dieses Vorgehen ist dann nützlich, wenn z. B. ein Produkt existiert, das als Ausgangspunkt für weitere Varianten dient. Die zweite Möglichkeit ist, dass ein solcher Kern nicht existiert und das Produkt ganz neu aufgebaut wird. Am Beispiel der eingangs erwähnten Codeverwaltung sind diese Möglichkeiten ebenfalls beide zu erkennen. Wird eine bestehende Datei mit Code modifiziert, so ist dies analog zur Anwendung eines Deltas auf ein Kernprodukt zu sehen. Wird hingegen eine neue Implementierung in die Codeverwaltung eingebracht, so gibt es naturgemäß keinen vorangegangenen Stand, auf den aufgebaut werden kann

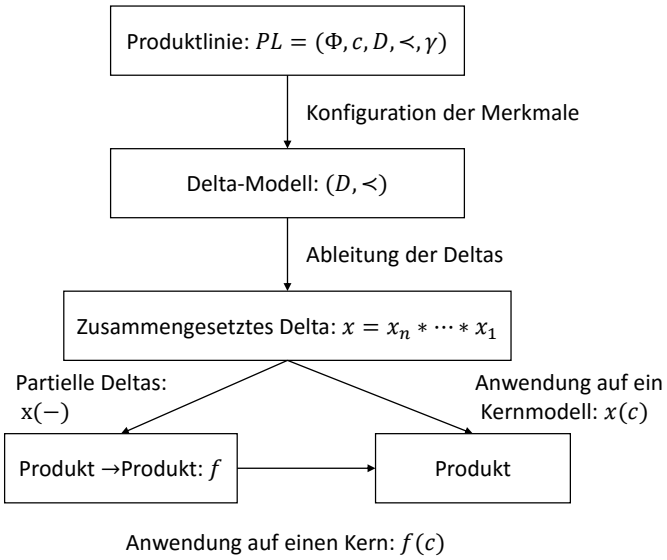


Abbildung 4.6: Beziehung zwischen Artefakten und Begriffen nach [CHS10].

und somit kein Kern- oder Vorprodukt. Eine formale Beschreibung der Delta-Modelle ist in [Sch10, CHS10] zu finden.

Bei der Anwendung von Delta-Operationen auf einen Ausgangszustand kann es zu Konflikten und Inkonsistenzen kommen. Beispielsweise wenn ein Element, das nicht vorhanden ist, modifiziert werden soll. Um solche Konflikte zu vermeiden und um eine bessere Übersichtlichkeit zu erlangen, werden normalisierte Delta-Modelle verwendet [Sch10]. Diese bestehen jeweils nur aus einem Typ von Delta-Operation, d. h. entweder aus add, mod oder rem. Aus jedem Delta-Modell oder jeder Kombination können drei normalisierte Delta-Modelle gebildete werden. Dies wird dadurch erreicht, dass die Delta-Operationen nach Typen sortiert abgearbeitet werden. Als erstes werden die Operationen zum Hinzufügen von Elementen angewendet. Im zweiten Schritt werden alle Modifikationen durchgeführt und anschließend die vorgesehenen Elemente gelöscht. Die normalisierten Delta-Modelle werden nacheinander angewendet. Im Anschluss an die Transformation der Modellelemente werden zunächst die neuen Verbindungen angelegt und anschließend die Operationen zum Löschen der Verbindungen durchgeführt. Es wird von der Annahme ausgegangen, dass die Delta-Operationen eines Deltas-Modells alle zeitgleich angewendet werden, d. h., es wird keine Anwendungsreihenfolge modelliert [Sch10].

Nach [Sch10] sind folgende Regeln für die Anwendung der Operationen vorgesehen:

- Ein Element oder eine Verbindung kann mehrfach hinzugefügt werden, taucht aber im Modell nur einmal auf.

- Wenn ein Element entfernt wird, werden alle Verbindungen, an denen es beteiligt ist, ebenfalls gelöscht.
- Die Operationen ein Element zu modifizieren oder zu löschen, das nicht existiert, ist nicht definiert.

Trotz Beachtung der Regeln können nach [Sch10] drei Arten von Konflikten auftreten: Das Anlegen und nachträgliche Löschen eines Elements oder einer Verbindung, das Modifizieren und anschließende Löschen von Elementen oder das Modifizieren und erneute Modifizieren von Elementen. Diese Anwendungen von Delta-Operationen sind grundsätzlich zulässig, weisen aber auf Inkonsistenzen innerhalb des Delta-Modells hin. Während der Transformation kann es passieren, dass die aktuell vorliegenden Modellelemente und Verbindungen kein valides Produkt im Sinne der zugrundeliegenden Architektur oder Sprache bilden (vgl. [HRRS11]). Erst nach Abschluss der Transformation wird ein valides Produkt erwartet. Eine formale Beschreibung der aufeinanderfolgenden Anwendung von Delta-Modellen einschließlich potentieller Konflikte und Wege zu deren Behebung ist in [CHS10] zu finden.

Die vorangegangene Betrachtung des Delta-Modells im Kontext der Software Produktlinien vermittelt einen Eindruck der Vielseitigkeit dieses Ansatzes. Grundsätzlich können die Modifikationen an Softwareprodukten mit diesem Ansatz beschrieben werden. Durch die Mächtigkeit der Transformation ist es möglich, jedes Produkt in ein komplett neues umzuwandeln. Im Extremfall wird das Ausgangsprodukt komplett entfernt und ein neues Produkt aufgebaut. Es ist möglich über die Produktlinie und die darin enthaltenen Features eine Klammer um die assoziierten Produkte zu bilden [CHS10]. Dies bedeutet, dass sie funktional gewisse Gemeinsamkeiten haben sollen. Diese Gemeinsamkeiten werden in [SRC+12] durch das Kernprodukt repräsentiert. An diesem Kernprodukt werden durch Delta-Modelle Veränderungen vorgenommen. Das Kernprodukt stellt für die von ihm anhängenden Produkte die gemeinsame Basis dar. Gleichwohl ist diese Herangehensweise durch eine Fokussierung auf den Lösungsraum geprägt, d. h., die Gemeinsamkeiten werden durch Features beschrieben. Eine Möglichkeit die Ähnlichkeit im Lösungsraum zu ermitteln und gegebenenfalls zu begrenzen, ist die Einführung eines Abstandsmaßes über die Delta-Operationen. Eine Übersicht über verschiedene Abstandsmaße ist in [BSG12] zu finden. Die Festlegung, welches Produkt Bestandteil einer Produktlinie bzw. Variante eines anderen Produkts ist, ist subjektiv [Lim94].

Man sieht, dass dieser intuitive Ansatz in vielen Anwedungsgebieten zum Einsatz kommen kann. Die erste Publikation der Delta-Modelle kam aus dem Umfeld der modellgetriebenen Softwareentwicklung [Sch10]. Ausgehend von diesem Ansatz entwickelte sich der Bereich des abstrakten Deltamodellierens [CHS10]. Darin wird die formale Basis für die Nutzung der Delta-Modelle gelegt. Dieses generische Konzept für die Entwicklung von Softwareproduktlinien wurde nachfolgend in verschiedenen Anwendungsbeispielen umgesetzt und erprobt. Im Folgenden werden einige Beispiele vorgestellt.

Die Delta-Modellierung wurde als deltaorientiertes Programmieren mit der Programmiersprache DeltaJava umgesetzt und erprobt [SBB+10]. Die Realisierung wurde mit einer featureorientierten Programmierung von Softwareproduktlinien verglichen. In diesem Rahmen wurde festgestellt, dass beide Ansätze gut skalieren, sich allerdings in einigen Punkten unterscheiden. Die Delta-Modelle sind in Bezug auf die Transformationsregeln sowie die Wahl des Startpunkts und damit der Anwendbarkeit auf bestehende Lösungen flexibler.

Durch die Möglichkeit, direkt aus Delta-Modellen Konflikte zu erkennen und partiell zu beheben, sind diese dem featureorientierten Programmieren voraus. Die Erweiterung der Funktionalität erfolgt im featureorientierten Programmieren durch ein Refactoring der bestehenden Produktlinie. Im Gegensatz dazu werden im deltaorientierten Programmieren weitere Delta-Module hinzugefügt.

Weitere Anwendungsbereiche sind die Nutzung von Delta-Modellen für die Beschreibung der Variabilität von Architektursprachen [HRRS11, HKR+11]. In diesen wird die Variabilität von Systemarchitekturen in Delta-Modellen beschrieben. Die Systemarchitektur wird ihrerseits durch die entsprechende Sprache ausgedrückt (z. B. MonitArc). Als weiterer Anwendungsfall wird ein Prozess für die Ableitung einer Delta-Sprache für eine gegebene textuelle Programmier- oder Modellierungssprache vorgestellt [HHK+13]. Der Prozess besteht aus Ableitungsregeln und Kontextbedingungen sowie einem Ablauf, der beispielhaft für eine textuelle Version der Statecharts durchgeführt wurde. Eine Anwendung von Delta-Modellen in der Praxis ist in [HMW12] beschrieben. Ausgehend von einem Workflow für die Entwicklung von Delta-Modellen wurde die Nutzung von Delta-Modellen in einem industrierelevanten Maßstab evaluiert. Eine Anwendung der Delta-Modelle im Kontext der Automatisierungstechnik ist in [KLL+14, KPST14] zu finden. Die Autoren beschreiben einen Ansatz, der auf einer multiperspektivischen Sicht beruht. Dabei werden drei verschiedene Sichten (Workflow, Architektur und Verhalten) auf ein Fertigungssystem genutzt, um dafür Delta-Modelle aufzustellen und daraus Steuerungscode zu generieren. Die Vorteile der Deltamodellierung werden in der einfachen Verständlichkeit des Ansatzes und der Flexibilität gesehen.

Ein Ansatz die Delta-Modellierung für Produktlinien zu verwenden wurde in [SSS17] vorgestellt. Kern des Konzepts ist ein Prozess für die Entwicklung von Lösungen. Dieser besteht aus einer kurzfristigen, einer mittelfristigen und einer langfristigen Phase. Es werden für die einzelnen Phasen jeweils Metriken vorgestellt, die eine Verbesserung der Implementierung ermöglichen. Über *Varianten Interfaces* können Varianten Punkte für eine eingeschränkte Transformation markiert werden. Ein sehr interessanter Ansatz ist die Verwendung von Delta-Modellen zur Transformation von anderen Delta-Modellen [LKS16]. Dieser Ansatz wird als Delta-Modell höherer Ordnung bezeichnet. Vorteil dieses Ansatzes ist, dass die vollständige Historie der Änderungen an Delta-Modellen abgebildet wird.

4.4 Modellierungsgrundlagen

Ziel von Modellen ist es, das System, das modelliert wird, besser zu verstehen [Kru04]. Die unterschiedlichen Arten von Modellen sind zahlreich wie ihre Anwendungsbereiche. In [VWB+09] sind folgende Beispiele für Modelle aus unterschiedlichen Anwendungsbereichen genannt:

- Kostenmodelle
- Rentenmodell
- Modelle für die Wettervorhersage
- Geometriemodelle

- Modelle für den Reglerentwurf
- Modelle für Börsenkurse

Genauso vielfältig wie die Verwendungsmöglichkeiten sind die Vorstellungen, was Modelle sind. Die unterschiedlichen Betrachtungsweisen stammen aus unterschiedlichen Anwendungsbereichen und haben sich über die Zeit hinweg entwickelt. Eine sehr generische Definition für Modelle ist in [DIN14] gegeben:

Definition 5 (Modell)**.** *Gegenstand, der es erlaubt Aussagen über einen anderen, modellierten Gegenstand zu treffen.*

Ausgehend von dieser Definition lassen sich Modelle als Abbildung eines Gegenstandes ansehen. Diese Abbildung wird für einen konkreten Zweck erstellt [BRS95]. Der Zweck gibt vor, in welcher Art und Weise die Sicht auf den Gegenstand verkürzt wird. Modelle sind deshalb verkürzt, da ein Gegenstand nicht in allen seinen Fassetten modelliert wird, sondern nur die für den jeweiligen Zweck relevanten Eigenschaften und Verhaltensweisen. Diese wird in der folgenden Definition von Modell in [IEC14a] aufgegriffen:

Definition 6 (Modell)**.** *Mathematische oder physikalische Darstellung eines Systems oder Prozesses, die das System oder den Prozess aufgrund bekannter Gesetzmäßigkeiten, einer Identifikation oder getroffener Annahmen genügend genau abbildet.*

Die beiden zitierten Definitionen unterstreichen mögliche unterschiedliche Blickwinkel auf Modelle. So werden diese, je nach persönlichem Hintergrund und dem konkreten Anwendungsfall anders definiert. Dies kann bei einer Zusammenarbeit von Personen aus unterschiedlichen Fachrichtungen und Branchen zu Missverständnissen führen. So versteht ein Regelungstechniker unter einem Modell grundsätzlich ein System von Differentialgleichungen, die mittels physikalischen Gesetzmäßigkeiten ein physisches System beschreiben. Hierbei handelt es sich um ein Modell im Sinne von Definition 5 und Definition 6. Ein technischer Modellbauer assoziiert mit einem Modell ein physisches Modell, beispielsweise von einem neuen Produkt. Dieses Modell lässt sich nur mit Definition 5 in Einklang bringen, da das Modell nicht mathematisch oder physikalisch ist. Aus demselben Grund erfüllt auch das Modell einer Anlagentopologie lediglich die Definition 5.

Für die vorliegende Arbeit dient Definition 7 aus [Epp08] als Grundlage. Sie beschreibt Modellsysteme als Erweiterung des Modellbegriffs, der ein Modell als verkürzte Abbildung eines physischen oder nicht-physischen Gegenstandes versteht. Dabei wird das Modell selbst als System verstanden, das aus internen Elementen besteht. Im Weiteren werden Modell und Modellsystem als Synonyme verwendet.

Definition 7 (Ein Modellsystem ist ein)**.** *Modell, das selbst als System strukturiert ist und das versucht den inneren Aufbau eines Systems so gut nachzubilden, dass im gewünschten Kontext und mit der geforderten Genauigkeit die äußeren Eigenschaften des Modellsystems mit denen des betrachteten Systems übereinstimmen.*

Da viele Modellierungssprachen objektorientiert sind [PGGS16], liegt der Fokus dieser Arbeit auf der Betrachtung von objektorientierten Modellen. Beispiele dafür sind das OPC UA Metamodell [IEC10], das Merkmalmodell [Mer12, Kam17] oder das allgemeine Prozedurmodell [NAM16, Sch16a].

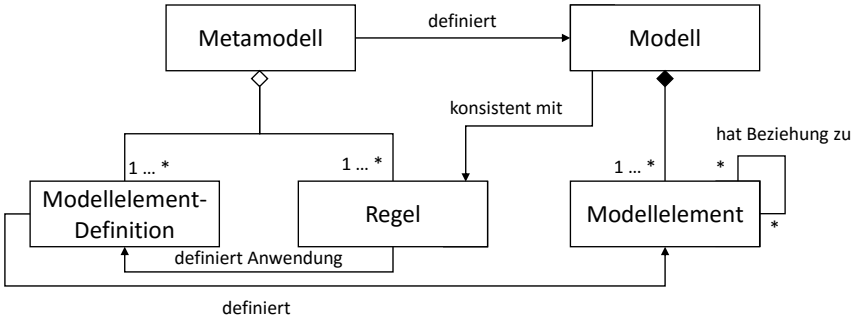


Abbildung 4.7: Formale Spezifikation des Metamodells nach [DIN14]

Eine Übersicht über die Grundsätze zur ordnungsgemäßen Modellierung ist in [BRS95] zu finden. Dies sind u. a. die Relevanz, die Klarheit und die Vergleichbarkeit eines Modells.

Im Folgenden werden zwei Klassifikatoren für Modelle vorstellt. Zum einen werden die verschiedenen Metamodell-Ebenen vorgestellt und ihre Bedeutung für Laufzeitumgebungen erläutert. Anschließend wird die Klassifikation nach den verschiedenen Sichten auf das modellierte System vorgestellt.

4.4.1 Ebenen der Modellierung - Metamodelle als Wegbereiter der Interoperabilität

Eine Möglichkeit, Modelle zu klassifizieren, ist die Einteilung über die Art des zu modellierenden Systems. Definition 7 folgend beschreiben Modelle reale Systeme. So kann beispielsweise ein Motor durch physikalische Gleichungen beschrieben werden. Diese Gleichungen folgen den Regeln der Physik. Die Physik stellt somit Regeln für die Modellierung von Gegenständen bereit und definiert Bestandteile des Modells (elektrischer Strom, Drehmoment). Sie ist nach Definition 8 aus [DIN14] ein Metamodell.

Definition 8 (Metamodell)**.** *Ein Metamodell definiert Aufbau und Bedeutung von Modellen*

In Abbildung 4.7 ist die formale Spezifikation des Metamodells dargestellt. Das Metamodell definiert Modelle und besteht aus Definitionen von Modellelementen und Regeln. Die Regeln eines Metamodells definieren die Verwendung der in diesem Metamodell definierten Modellelemente. Das nach den Vorgaben eines Metamodells aufgestellte Modell ist konsistent mit den Regeln des Metamodells und verwendet die definierten Modellelemente (vgl. [BRS95]). Der Begriff *Modellelement* kann sehr sehr allgemein verstanden werden. So können Modellelemente, je nach Art des betrachteten Modells, Gleichungen oder auch Objekte (z. B. in einem Strukturmodell) sein. Der Zusammenhang zwischen einem Metamodell und dem Modell ist deutlich stärker als der Zusammenhang zwischen einem Modell und dem modellierten System. Das Modell bildet üblicherweise nur einen Teilaspekt des

modellierten Systems ab, wohingegen das Metamodell das Modell in Gänze beschreiben muss.

Es ist möglich, Metamodelle, d. h. ihre Modellelemente und Regeln, ohne Abhängigkeiten zu anderen Modellen zu definieren. Eine andere Herangehensweise ist, Metamodelle nach den Vorgaben anderer Metamodelle aufzustellen. Die Physik, die in dem Beispiel die Vorgaben zur Beschreibung des Motors liefert, basiert auf der Mathematik. Die Mathematik ist in diesem Kontext ein Metametamodell, d. h., sie ist die Grundlage für die Definition eines Metamodells (der Physik).

Wie an diesem einfachen Beispiel mit der Mathematik, der Physik und dem Motor gezeigt, lassen sich Metamodelle in einer Hierarchie verwenden. Ausgehend von Metametamodellen (Mathematik) werden Metamodelle (Physik) definiert, mit denen Modelle von Gegenständen entworfen werden. Analog dazu werden auch in anderen Bereichen (z. B. in der objektorientierten Programmierung [OMG16]) Metamodelle verwendet. Es gibt keine Vorgaben, wie viele Hierarchiestufen verwendet werden müssen. Es sind, ausgehend von einer zweistufigen Architektur bestehend aus Klasse und Instanz, beliebig viele Ebenen möglich [OMG16]. Allerdings nutzen die meisten Systeme weniger als fünf Ebenen (Metametamodell, Metamodell, Modell, Instanz).

Der Nutzen eines Metamodells entsteht aus der einheitlichen Verwendung der Modellelemente und Regeln, die das Metamodell definiert. Dieser Nutzen wird gesteigert, wenn ein Metamodell von möglichst vielen Modellerstellern als richtig angesehen und verwendet wird. Auch hier kann die Mathematik als Beispiel herangezogen werden. Überall auf der Welt nutzen Menschen Regeln und Elemente der Mathematik. Dabei ist es irrelevant, ob der Anwendungsfall in der theoretischen Physik oder im Bereich der Kostenrechnung liegt, die Regeln sind immer gleich. Je nach Anwendungsfall kann jedoch der Umfang der genutzten Regeln und Modellelemente schwanken. Während möglicherweise in der Kostenrechnung die vier Grundrechenarten ausreichen, kann in der theoretischen Physik die Verwendung von sehr speziellen Regeln und Modellelementen nötig sein.

Im technischen Kontext wurde die Notwendigkeit für die Verwendung von gemeinsamen Metamodellen für Interoperabilität festgestellt [PSU+14]. Es wird zwischen *Referenzmodellen* und *Kernmodellen* unterschieden. Kernmodelle sind analog zu physikalischen Gesetzmäßigkeiten immer gültig, ganz gleich, ob sie explizit oder implizit angewendet werden. Eine Sammlung von Kernmodellen ist in der DIN SPEC 40912 [DIN14] normiert. Ein Beispiel für ein Kernmodell ist das Lebenszyklusmodell, das beschreibt, was der Lebenszyklus einer Entität ist. Dieses Modell ist so generisch, dass es mindestens implizit verwendet wird, immer dann, wenn ein Lebenszyklus modelliert oder verwendet wird. Im Gegensatz zu den Kernmodellen haben Referenzmodelle nur einen hinweisenden Charakter. Sie besitzen keine grundsätzliche Gültigkeit, sondern sind ausschließlich in einem Anwendungsbereich gültig. In diesem Anwendungsbereich ist z. B. die Sammlung von Entwurfsmustern für objektorientierte Softwarearchitekturen zu sehen [GHJV11]. Diese Sammlung beschreibt in der objektorientierten Programmierung häufig auftretende Konstrukte, die für einen Programmierer nützlich sein können, aber nicht alternativlos sind.

Eine wesentliche Voraussetzung für die Entwicklung und Nutzung von gemeinsamen Modellen ist die Erlangung eines gemeinsamen Verständnisses aller relevanten Begriffe. Die „klare, konsistente und allgemein anerkannte textuelle Beschreibung von Begriffen“ ist

eine Voraussetzung für die Entwicklung von Referenzmodellen im Rahmen von Industrie 4.0 [PSU+14]. Typischerweise treten bei der Verwendung von unterschiedlichen Begriffsdefinitionen in einer Diskussion zwei Effekte auf: Entweder verwenden die Beteiligten unterschiedliche Begriffe für den gleichen Gegenstand oder sie verwenden den gleichen Begriff für unterschiedliche Gegenstände [WGE+17]. Durch die Entwicklung und durchgängige Verwendung von Referenz und Kernmodellen werden diese Schwierigkeiten überwunden und die Entwicklung von neuen Lösungen unterstützt.

Verwendung von (Meta-)Metamodellen in der Automatisierungstechnik

Nach der Vorstellung von Modellen und Metamodellen folgt in diesem Abschnitt die Beschreibung, wie diese in der Automatisierungstechnik verwendet werden. Neben den Anstrengungen, Metamodelle aufzustellen und anschließend durch Abstimmung mit Partnern als Referenzmodelle bzw. Kernmodelle zu verwenden, besteht der Wunsch die Modelle in konkreten Anwendungsfällen einzusetzen. Der Ansatz, (Meta-)Modelle im Engineering von Anlagen zu verwenden und damit ein durchgängiges modellbasiertes Engineering zu verwirklichen, wird in [VHDF+14] skizziert. In [BSF+09] beschreiben die Autoren, wie aus Strukturmodellen von Anlagen automatisiert Simulationsmodelle erzeugt werden können. Dabei kommt ein xml-basiertes Austauschformat für die Anlagenstrukturmodelle zum Einsatz. Ein Metamodell für die Beschreibung von Abläufen in der Automatisierungstechnik wird in [YGE13] vorgestellt. Für die Erstellung von Automatisierungslösungen sind in [WKS+16] zwei Wege beschrieben, die auf der Anwendung von Modellen basieren. Zum einen ist die modellbasierte Codegenerierung beschrieben, bei der die Problemstellung und z. B. die Anlage in Modellen beschrieben sind. Daraus wird dann die Automatisierungslösung generiert. Zum anderen kann die Lösung durch Modellzugriffe und Transformationen zur Laufzeit erstellt werden. Der erste Weg fügt sich hervorragend in bestehende Engineeringprozesse ein, wohingegen der zweite Ansatz den Vorteil hat, dass Modelle und Algorithmen, die auf den Modellen operieren, getrennt sind. Daraus folgt eine sehr gute Wiederverwendbarkeit von Lösungen für andere Anwendungsfälle. Zur Umsetzung dieser modellbasierten Lösungen ist in jedem Fall eine Vorstellung erforderlich, wie das konkrete Modell aufgebaut ist.

Neben der Definition von Metamodellen muss geklärt werden, wie diese zu speichern und zugänglich zu machen sind. Damit eng verknüpft ist die Frage, wie die Modelle, die unter Verwendung der Metamodelle entwickelt werden, genutzt werden. Sinnvollerweise sind die Metamodelle während der Verwendung der aus ihnen aufgebauten Modelle ebenfalls verfügbar. Die Autoren stellen in [WKS+16] drei grundsätzliche Möglichkeiten dar, Modelle zu verwenden: dateibasiert, in Datenbanken und in Laufzeitumgebungen. Die Ansätze unterscheiden sich hinsichtlich des Zugriffs auf die Modelle (Interpreter für Dateien, Queries bei Datenbanken und Dienste bei Laufzeitumgebungen) und der Möglichkeit dynamische Modelle zu verwalten. Daraus resultiert, dass der Datei- und der Datenbank-basierte Ansatz eher für das Engineering und weniger für die Nutzung der Modelle zur Laufzeit geeignet sind. Die Laufzeitumgebung ist darüber hinaus für die dynamische Verwendung von Modellen zur Laufzeit geeignet.

Bei der Realisierung von Modellen in Laufzeitumgebungen ist das der Laufzeitumgebung und den Metamodellen zugrundeliegende Metametamodell relevant. So stellt die OMG

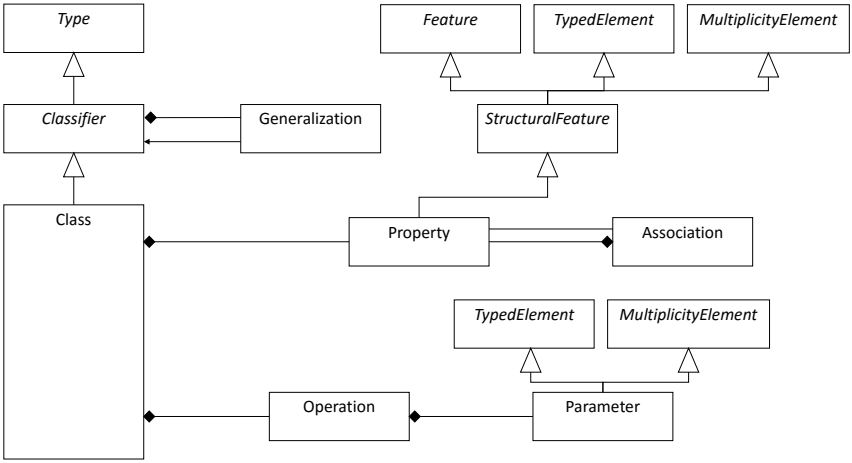


Abbildung 4.8: Vereinfachte Darstellung des Metametamodell der OMG [OMG16]

mit der Meta object facility ein Metametamodell für die Erstellung von Modellen bereit [OMG16]. OPC UA [IEC10] definiert ein Metametamodell als Grundlage für die Objektstrukturen in einem OPC UA Server. In Laufzeitsystemen wie FASA [WGKO15] oder der Laufzeitumgebung des Lehrstuhls für Prozessleittechnik [Alb03] kommen ebenfalls eigene Metametamodelle zum Einsatz.

Es ist zu erkennen, dass die Metametamodelle und die Metamodelle für die Interoperabilität relevant sind. Wenn sie kompatibel sind, ermöglichen sie auf Modellebene Interoperabilität und Austauschbarkeit. Allerdings erfordert eine Integration von verschiedenen Laufzeitumgebungen in einem Anwendungsfall, die ausschließlich im Verbund gelöst werden kann, ein Verständnis für die zugrunde liegenden Metamodelle der jeweiligen Laufzeitumgebungen. Ein Beispiel dafür ist die Darstellung der Objektstruktur der Laufzeitumgebung des Lehrstuhls für Prozessleittechnik als OPC UA Nodestore. Grundsätzlich ist diese Abbildung unproblematisch, allerdings ist das Metametamodell der Laufzeitumgebung sehr stark an der Containment-Beziehung zwischen Objekten orientiert. Das OPC UA Metametamodell kennt diese Art der Beziehung auch, allerdings ohne ihr die Priorität beizumessen. Dieses einfache Beispiel verdeutlicht, dass nicht nur die zugrundeliegenden Metamodelle, sondern auch die (implizit) verwendeten Metametamodelle für die Interoperabilität relevant sind.

Metametamodelle

Im Folgenden werden drei Beispiele für Metametamodelle vorgestellt: Die Meta Object Facility der OMG, das OPC UA Metamodell und das Metametamodell der Laufzeitumgebung des Lehrstuhls für Prozessleittechnik. In Abbildung 4.8 ist das vereinfachte Metametamodell der OMG dargestellt. Kern des Modells ist *Class*, die sich aus Eigenschaften (*Property*) und Operationen (*Operation*) zusammensetzt. *Class* ist von *Classifier* abgeleitet, und *Clas-*

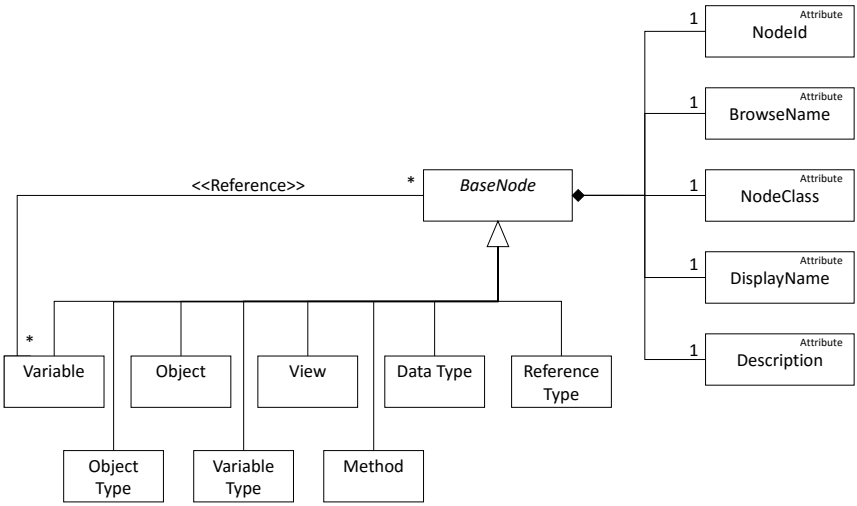


Abbildung 4.9: OPC UA Metamodell nach [LM06]

sifier wiederum ist von *Type* abgeleitet. Zweck des Modells ist es, eine Grundlage für die Beschreibung von Softwaremodellen zu schaffen. Des Weiteren besitzt es breite Akzeptanz in der Softwareentwicklung [SRVK10]. Die OMG hat dieses Metametamodell als Grundlage für darauf aufbauende Metamodelle standardisiert und schafft so eine einheitliche Vorstellung, wie objektorientierte Modelle im Kontext von UML 2.0 aufgebaut werden.

Ein Metametamodell, das in der Automatisierungstechnik aktuell viel diskutiert wird, ist das OPC UA Metamodell (vgl. Abbildung 4.9). OPC UA [IEC10] definiert zwei Komponenten: Ein Kommunikationsprotokoll und einen objektorientierten Modellspeicher (Nodestore). Das Kommunikationsprotokoll spezifiziert die Zugriffe auf den Nodestore. Im Wesentlichen handelt es sich dabei um Services zum Lesen und Schreiben von Werten sowie dem Erzeugen und Löschen von Objekten. Das in Abbildung 4.9 dargestellte Modell bildet die Grundlage für den OPC UA Namensraum. Der Kern dieses Modells ist der *BaseNode*, der sich aus *NodeId*, *BrowseName*, *NodeClass*, *DisplayName* und *Description* zusammensetzt. Jedes Objekt in OPC UA ist von BaseNode abgeleitet und kann Referenzen zu Variablen haben. Sowohl *Variable* als auch *Object* können über die entsprechenden Objekte *Object Type* und *Variable Type* typisiert werden. Neue Datentypen können mittels *Data Type* aus den vorhandenen Datentypen zusammengesetzt werden. Darüber hinaus ist es mit *Reference Type* Objekten möglich, zusätzliche Arten von Referenzen zu definieren. Ausführbare Methoden werden über *Method* Objekte realisiert.

Die erzeugte Objektstruktur, d. h. alle Objekte in einem Nodestore, kann zur Laufzeit erkundet und interpretiert werden. Die Kenntnis des Metamodells ist dabei unerlässlich, um die Semantik der Objekte korrekt interpretieren zu können (z. B. eines Method-Objektes).

Grundlage der am Lehrstuhl für Prozessleittechnik entwickelten Laufzeitumgebung

ACPLT/RTE ist das in Abbildung 4.10 dargestellte Metametamodell. Kern des Modells ist *object*, von dem alle anderen Teile des Modells abgeleitet sind. Es ist zu erkennen, dass ähnlich wie im OPC UA Metamodell sowohl Operationen als auch Variablen eigene Objekte sind. Die Klassen, die in der Laufzeitumgebung instanziiert werden können, sind selbst Instanzen der Metaklasse *class* und werden im System explizit verwaltet. Kern des Modells ist der Ringschluss zwischen class und object, durch den alle Objekte Instanzen ihrer Klassen werden und die Laufzeitumgebung sich selbst beschreibt [WKS+16].

Beim Vergleich des OPC UA Metamodells mit dem ACPLT-Metametamodell fällt die angesprochene Fokussierung auf die containment Beziehung zwischen object und *Domain* auf. Damit wird erzwungen, dass sich jedes Objekt, unabhängig von seinem Assoziationsnetzwerk, mit anderen Objekten in eine namensgebende Grundbaumstruktur einordnet. Diese Namensbaumhierarchie gibt es in OPC UA Informationsmodellen nicht. Trotz der leichten Unterschiede zwischen den Metametamodellen können die mit ihnen entwickelten Modelle ineinander überführt werden [SRVK10]. So können durch die Transformation der Informationsmodelle Datenbanken über OPC UA zugänglich gemacht werden [GPE16].

Die Kenntnis von Metametamodellen und deren korrekter Verwendung werden in einer zusammenwachsenden Landschaft von Laufzeitumgebung und Modellen immer wichtiger. Ein Indikator dafür ist die zunehmende Fokussierung auf das OPC UA Metamodell auch gerade im Kontext von Industrie 4.0. Hierbei spielen Metamodelle eine wichtige Rolle. Bei der Umsetzung der wandelbaren Fabrik erkunden Geräte andere Geräte und führen mit diesen gemeinsame koordinierte Aktionen durch. Dafür reicht es nicht, dass Geräte auf der Ebene von Kommunikationsprotokollen kompatibel sind, sondern sie müssen darüber hinaus in der Lage sein, gegenseitig die Informationsmodelle zu interpretieren. Dafür sind gemeinsame Metamodelle sowohl auf der Seite des Senders als auch des Empfängers unerlässlich.

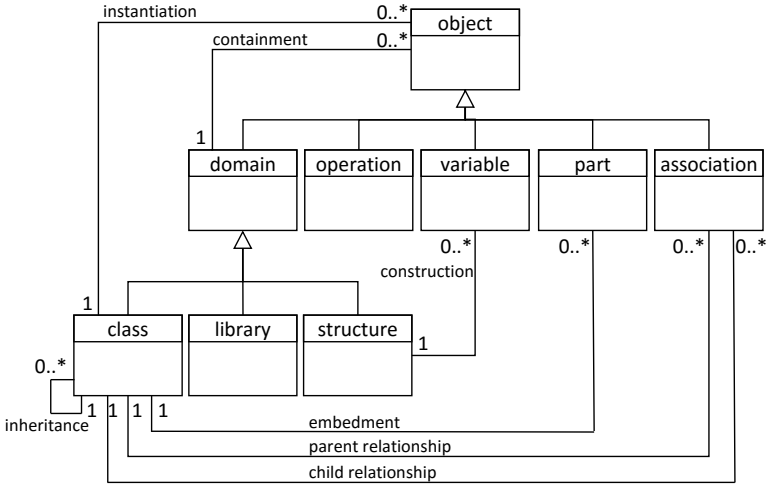


Abbildung 4.10: Metametamodell der Laufzeitumgebung ACPLT/RTE [WKS+16]

4.4.2 Modellierungssichten

Bei der Modellierung ist es wichtig, sich darüber im Klaren zu sein, welche Aspekte eines Gegenstandes modelliert werden sollen. Handelt es sich um mehrere Aspekte, stellt sich zusätzlich die Frage, ob diese Aspekte in einem Modell abzubilden sind, oder ob eine Unterteilung in mehrere Modelle nicht sinnvoller ist. Im folgenden Abschnitt wird auf diese Fragen eingegangen und es werden Ansätze präsentiert, welche Sichten eines Gegenstands in unterschiedlichen Modellen dargestellt werden sollten.

In der Automatisierungstechnik ist diese Trennung von Aspekten ein bekannter Ansatz. So ist beispielsweise in AutomationML [IEC14c, DLPH08] die Trennung der Informationen über einen betrachteten Gegenstand in drei Modelle vorgesehen:

- **Struktur**
  Aufbau und Beschreibung von Anlagen mit CAEX nach [IEC16]
- **Geometrie und Kinematik**
  Beschreibung der Geometrie und der Kinematik von z. B. Robotern. Als Format wird COLLADA verwendet.
- **Logik**
  Beschreibung von Schrittketten und Bausteinnetzwerken mit PLCopen. PLCopen definiert ein neutrales Austauschformat für die industrielle Leittechnik auf Basis von xml.

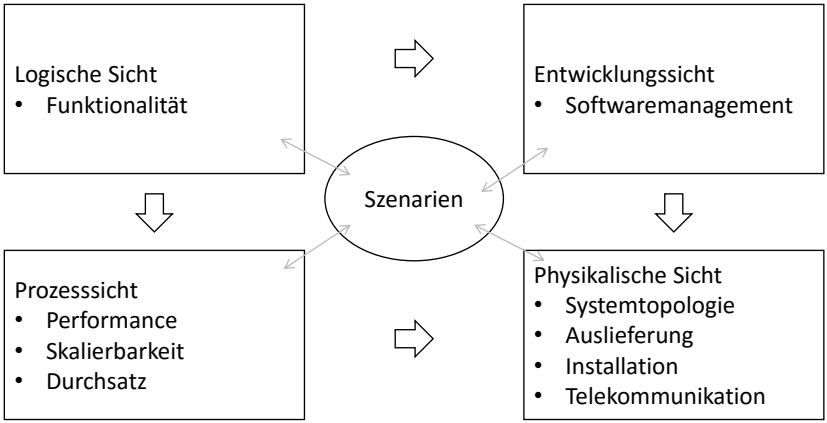


Abbildung 4.11: 4+1 Sichten Modell nach [Kru95]

Es ist zu erkennen, dass die Trennung entlang der verschiedenen Gewerke in einer Anlage vorgenommen wird: Die mechanische Modellierung, die Modellierung aus Sicht der Automatisierungstechnik und die Analgenstruktur. Dieser Ansatz ist weit verbreitet, da historisch gesehen die Tools so entstanden sind, wie sie von den unterschiedlichen Fachrichtungen benötigt wurden. So gibt es beispielsweise eigene Tools für die Prozessplanung (z. B. Verfahrenstechnik), für das Engineering der Automatisierungslösung und für die Bedienung der Anlage [WGE+17]. Jedes dieser Tools bildet eine eigene Sicht auf die gleiche Anlage ab und verwendet zur Beschreibung eigenen Metamodelle. Die Herausforderung in der Automatisierungstechnik besteht darin, alle Sichten auf die Anlage zu integrieren und mehrfache (mglw. inkonsistente) Datenhaltung zu vermeiden.

In der Softwarearchitektur gibt es ebenfalls Konzepte, die unterschiedlichen Sichten von modellierten Gegenständen zu trennen. In [Kru95] wird das 4+1 Sichten Modell für die Softwarearchitektur vorgestellt. In Abbildung 4.11 ist das Modell dargestellt. Im Mittelpunkt stehen die unterschiedlichen Szenarien, die für die Entwicklung einer Softwarearchitektur relevant sind. Die logische Sicht betrachtet die Funktionalität des Softwaresystems. In der Regel ist das die Sicht, die ein Nutzer auf das System hat. Hierbei stehen die funktionalen Anforderungen im Vordergrund. Aus der logischen Sicht ergeben sich die Entwicklungssicht, die für den Softwareentwickler relevant ist, und die Prozesssicht für den Systemintegrator. In der Entwicklungssicht wird beschrieben, wie die Softwaremodule organisiert werden. Nicht-funktionale Anforderungen wie die Verteilung der Softwareartefakte oder die Fehlertoleranz werden in der Prozesssicht adressiert. Alle anderen Sichten münden in der physikalischen Sicht, die den tatsächlichen Aufbau des Gesamtsystems aus Soft- und Hardware sowie der Kommunikation beschreibt.

Um die verschiedenen Sichten erstellen und bearbeiten zu können, stehen verschiedene grafische Modelle zur Verfügung. Diese enthalten Diagramme für Szenarien, Anwendungsfälle,

Klassen, Deployment, Zustände und Komponenten [Kru04]. Ein Beispiel für eine derartige Sammlung von Diagrammen ist die Modellierungssprache UML [ISO12]. Diese definiert sowohl Struktur- als auch Verhaltensdiagramme, deren Elemente gemischt in einem Modell verwendet werden können.

In [FR07] betonen die Autoren die Bedeutung davon, die verschieden Sichten eines Softwaresystems während seines Entwurfs zu berücksichtigen und zu modellieren. Für die Automatisierungstechnik bedeutet der Ansatz, beispielsweise die funktionale Sicht von der Implementierung und der Verteilung auf verschiedene Hardwareplattformen zu trennen, jedoch keinesfalls die Abhängigkeiten zu vernachlässigen.

4.4.3 Modelle in der Automatisierungstechnik

In folgenden Kapiteln werden kurz die für diese Arbeit relevanten Modelle der Automatisierungstechnik vorgestellt. Die Relevanz ergibt sich entweder durch ihre direkte Verwendung oder durch ihren Vorbildcharakter für die Lösung von Problemstellungen. Zunächst wird das grundlegende und vielseitig einsetzbare Merkmalmodell vorgestellt. CAEX und seine Spezialisierung R&I Fließbilder für die Prozessindustrie wird anschließend als Beispiel für ein Rollenmodell vorgestellt. Abschließend wird ein Modell für die Beschreibung der Sprachen des Engineerings erläutert.

Merkmale

Merkmale beschreiben charakterisierende Eigenschaften von Dingen, die im Betrachtungszeitraum als konstant angesehen werden können [EE13]. Abzugrenzen sind Zustände, die einer fortlaufenden Änderung unterworfen sein können. Die Herausforderung im technischen Kontext besteht darin, Merkmale so zu definieren und zu modellieren, dass bei der Interpretation eines Wertes klar ist, was dessen Semantik ist. Es muss also aus der Verknüpfung eines Wertes mit einer Merkmaldefinition hervorgehen, welche Semantik dieser Wert hat. Beispielsweise wird beschrieben, dass es sich bei diesem Wert um den Durchmesser eines Rohres handelt und ob es der Innen-, Mittel- oder Außendurchmesser ist.

Das Merkmalmodell nach [EMPA17] lässt sich in drei Bestandteile unterteilen: Der beschriebene *Merkmalträger*, das semantische System und das Anwendungssystem. Der Merkmalträger besteht aus den spezifischen Merkmalen, deren Werte unbekannt sein können, da sie nur im Rahmen der Messungenauigkeiten messbar sind. Das semantische System besteht aus einem Modell der Merkmalträger, den Merkmalträgertypen und der Beschreibung der Merkmale. Das semantische System beruht auf globalen Standards wie z. B. der IEC 61360 [IEC17]. Bekannt ist auch ecl@ss, ein zentrales Repository für Definitionen von Merkmalen und den korrespondierenden Merkmalträgertypen. Optimalerweise legen semantische Systeme für alle Anwendungssysteme die Semantik fest, sodass Informationen zwischen diesen ausgetauscht werden können.

Innerhalb der Anwendungssysteme werden die abgebildeten Merkmalträger durch Ausprägungsaussagen beschrieben (vgl. [EMPA17]). Dies sind Werte mit einer Referenz auf ein Merkmal, einer Einheit, einer Erzeugungszeit, einer Quelle, einer Aussagesemantik und

einer logischen Einordnung. Über die Aussagesemantik kann die Aussage entweder als Anforderung, Zusicherung, Messwert oder Sollwert charakterisiert werden. Mit der Aussagelogik wird festgelegt, ob ein festgelegter Wert über- oder unterschritten werden soll. Alternativ kann die Gleichheit zu einem bestimmten Wert festgelegt werden. Die Referenz auf das Merkmal, zu dem eine Aussage gemacht wird, ermöglicht einem Nutzer des Wertes, diesen entsprechend seiner Bedeutung zu interpretieren. Dies ist der Schlüssel zur Interoperabilität zwischen Systemen.

Ein aktuelles Anwendungsgebiet von Merkmalsausprägungsaussagen ist die Modellierung von Verwaltungsschalen [PE17]. In diesem Kontext werden Merkmale genutzt, um die Semantik von Werten allgemeingültig beschreiben zu können und damit die Interoperabilität zu steigern [EMPA17]. Dafür wird die ursprüngliche Definition eines Merkmals weiter gefasst und auch Zustände und Parameter durch Merkmalsausprägungsaussagen ausgedrückt. In [Mer12, EE13] wird die Nutzung von merkmalbasierten Informationen für verschiedene Anwendungen vorgestellt. Dort sind weiterführende Informationen zu den Merkmalmodellen sowie ihrer Anwendung zu finden.

Entity-Relationship-Systemmodell

Das Entity-Relationship-Systemmodell (ER-Systemmodell) ist ein Kernmodell [DIN14] und als solches ein Metamodell für die Beschreibung von Systemen. Das ER-Systemmodell kann zur Beschreibung aller Arten von imaginären oder physischen Systemen verwendet werden. Die Systeme werden als Netzwerk aus miteinander in Beziehung (Relationship) stehenden Elementen (Entitys) beschrieben. Systemelemente können sowohl Klassen als auch Objekte sein. Das Systemmodell ist so flexibel einsetzbar.

Grundannahme des Modells ist, dass ein System eine Hülle, d. h. eine Systemgrenze besitzt und aus Elementen aufgebaut ist. Das System ist von außen betrachtet ein Element, in seinem Inneren ist es jedoch aus einer Struktur von verbundenen Elementen aufgebaut. Diese Elemente können wiederum einen internen Aufbau besitzen und so aus weiteren Elementen zusammengesetzt sein. Eine weitere Annahme ist, dass die Systemhülle nur an vordefinierten Stellen überwunden werden kann.

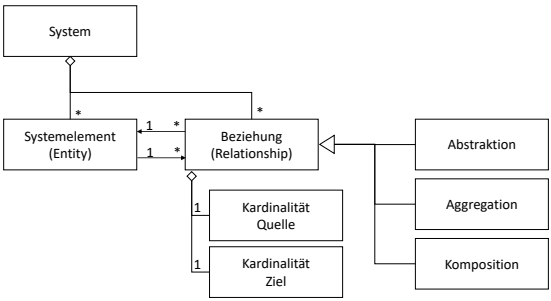


Abbildung 4.12: ER-Modell nach [DIN14]

In Abbildung 4.12 ist das Systemmodell dargestellt. Es ist zu erkennen, dass sich Systeme aus *Systemelementen* und *Beziehungen* zusammensetzen. Sie sind über *Quell-* und *Zielkanten* miteinander verbunden. Die Beziehung ist eine 1:1 Verbindung, jedoch können die Systemelemente beliebig viele Verbindungen untereinander besitzen. Beziehungen sind klassifiziert. Im Kernmodell sind die *Abstraktion*, die *Aggregation* und die *Komposition* vorgesehen. Diese können um zusätzliche Klassen ergänzt werden. Zusätzlich kann die Anzahl der zu verbindenden Systemelemente durch *Kardinalitäten* definiert werden.

Systemelement-Interface-Connection Modell

Das Systemelement-Interface-Connection (SIC) Modell ist ein Metamodell für die Beschreibung von Systemen und gehört zu den Kernmodellen (vgl. [DIN14]). Die Grundannahmen im Hinblick auf den Aufbau von Systemen sind die Gleichen wie für ER-Systemmodelle.

Der Aufbau des SIC-Modells ist in Abbildung 4.13 zu erkennen. Kern des Modells ist das Systemelement. Ein System besteht aus Basiselementen. Das System und das Systemelement bestehen jeweils aus Schnittstellen. Das Modell ermöglicht so die Beschreibung eines hierarchisch aufgebauten Systems. Die Verbindungen zwischen den Schnittstellen werden durch Verbindungen modelliert.

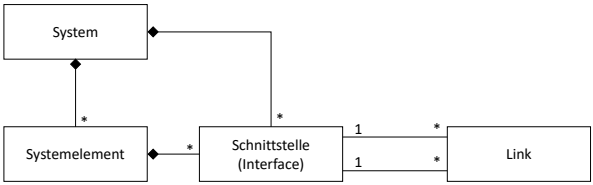


Abbildung 4.13: SIC-Modell nach [DIN14]

Anwendungsbeispiele, in denen das SIC-Modell benutzt wird, sind die Funktionsbausteinnetzwerke oder R&I Fließbilder. Das Funktionsbausteinnetzwerk stellt ein System dar, das aus Funktionsbausteinen (Basiselementen) und anderen Funktionsbausteinnetzwerken (Systemen) zusammengesetzt ist. Die Netzwerke und Bausteine stellen ihre Schnittstellen nach außen über Ports (Interfaceelemente) dar. Die Ports sind vergleichbar mit den Interfaces im SIC-Modell.

Modell für Sprachen des Engineering

Ein Metamodell für das Engineering von Funktionsbausteinsprachen wird in [WGE+18] vorgestellt. Ausgangspunkt dieses Beitrags war die Heterogenität der Laufzeitsysteme innerhalb des BaSys 4.0 Projekts. So entstand die Idee, ein Metamodell zu entwickeln, das von den jeweiligen Sprachen abstrahiert und die Verwendung einer einheitlichen Schnittstelle für das Engineering möglich macht. Das Metamodell ist in Abbildung 4.14 dargestellt.

Grundlage der Überlegungen ist das SIC-Modell. Analog zu diesem wurden die verwendeten Arten von Bausteinen in komplexe und einfache Bausteine unterteilt. Einfache Bausteine

werden als atomar betrachtet und können ihrerseits zu komplexen Bausteinen zusammengesetzt werden. Jeder Baustein verfügt über ein Zustands- und ein Ausführungsmodell. Jedes dieser Modelle stellt eine Schnittstelle für die Interaktion bzw. den Datenzugriff bereit. Alle Bausteine verfügen über eine Schnittstelle zur Interaktion mit der Umgebung. Diese besteht aus Signalports und Serviceschnittstellen. Signalports können nach Eingang, Ausgang oder bidirektional typisiert werden. Verbindungen zwischen Signalports werden durch Links und Verbindungsobjekte dargestellt. Innerhalb der Verbindungsobjekte werden z. B. die Konvertierung von Werten oder deren Übertragung realisiert.

Eine spezielle Art eines komplexen Bausteins ist die Prozedur im Sinne der NE 160 [NAM16]. Diese besteht aus Schritten und Transitionen, die jeweils miteinander zu einer Schrittkette verbunden sind. Innerhalb der Transition werden die Bedingungen für deren Schalten durch logische Gleichungen dargestellt. Schritte können entweder elementare Schritte, Makro-Schritte oder Parallelisierungen sein. Elementare Schritte bestehen aus Aktionsaufrufen, die beispielsweise das Setzen von Werten oder das Aufrufen eines Dienstes sind. Sowohl die Makro-Schritte als auch die Parallelisierungen sind jeweils Teilprozeduren und werden als abgeleitete Prozeduren modelliert.

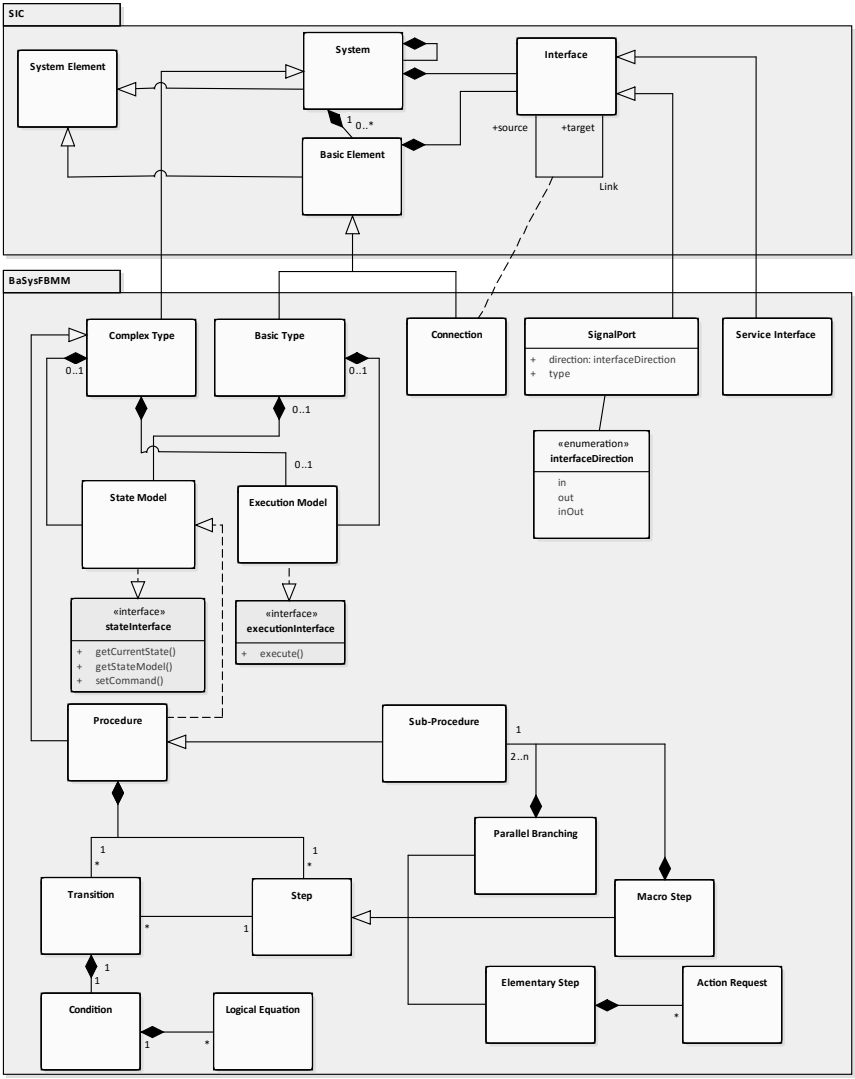


Abbildung 4.14: BaSys Metamodell für Engineeringsprachen nach [WGE+18]

4.5 Diskussion des Stands der Wissenschaft

In diesem Kapitel wurde der Stand der Wissenschaft zur Wiederverwendung vorgestellt. Es wurde gezeigt, dass dieses Thema in der Informatik weiterhin Gegenstand von Arbeiten ist. Auch in der Automatisierungstechnik werden zunehmend Anstrengungen unternommen, die Wiederverwendung von bestehenden Lösungen zu unterstützen. Durch den zunehmenden Einsatz von Laufzeitsystemen mit komponentenorientierten Metamodellen rückt die Wiederverwendung von Komponentensystemen in den Fokus. Durch andere Arbeiten im Bereich der wandelbaren Fabrik, wie die Verteilung von Applikationen und die Änderung von Funktionalität zur Laufzeit, wird dieser Trend unterstützt.

Eine Verbesserung der Wiederverwendung ist in der Automatisierungs- und Softwaretechnik gleichermaßen ein präsentes Thema. Das Konzept aus [ODU13] ist aufgrund der in Kapitel 3 geforderten Anwendbarkeit für hybride Systeme (Anforderung R13) für den in dieser Arbeit skizzierten Anwendungsfall ungeeignet. Die Beschreibung von AT-Modulen in [Mah14] ist eine funktionale Betrachtung zu hybriden Modulen. Das vorgestellte Konzept zielt nicht auf die Behandlung von entstehenden Varianten ab. Allerdings bilden die Module eine Ausgangsbasis für die Entwicklung von Wiederverwendungsgegenständen.

Die Berücksichtigung der Variabilität von Produkten zur Unterstützung der Wiederverwendung hat sich in anderen Domänen wie der Softwaretechnik und dem Maschinenbau als sehr zweckmäßig herausgestellt. Sie sind für den hier betrachteten Anwendungsfall auch im Fokus. Auf Produktlinien basierende Ansätze werden mit großem Erfolg für die Lösung vieler Aufgaben eingesetzt. Die kompositionellen und annotativen Verfahren sind für die Anwendung in bestehenden Systemen nur bedingt geeignet (R3). Im Gegensatz zu Delta-Modellen verfügen sie weder über die Flexibilität, Artefakte zu einer Lösung hinzuzufügen, noch sie wieder entfernen zu können [Sch18]. Für die betrachteten, sich evolutionär entwickelnden Systeme ist eine vollständige Transformation besser geeignet als die kompositionellen und annotativen Verfahren. Der zeitliche und monetäre Aufwand sind die Hindernisse für den Einsatz des auf Varianten basierenden Konstruierens [VHON18]. Die neue Delta-Modellierung ist besser für den Einsatz in bestehenden Lösungen geeignet (R3). Eine modellbasierte Beschreibung der Variabilität, wie in (R2) gefordert, ist zudem auch möglich. Delta-Modelle sind ein intuitiver Weg, den Unterschied zwischen zwei Produkten bzw. zwei Komponentensystemen zu beschreiben. Sie können flexibel auf unterschiedliche Anwendungsfälle angewendet werden. Es ist möglich, nicht nur einzelne Komponenten und Verbindungen, sondern ganze Teil-Systeme in einen neuen Anwendungsfall zu übernehmen.

Es wurde gezeigt, dass modellbasierte Lösungsansätze in der Automatisierungstechnik verbreitet sind und als zukunftsträchtig angesehen werden. Einheitliche Metamodelle sind für die Interoperabilität aber auch für das gemeinsame Verständnis der Systeme relevant. Dies gilt auf der Ebene der Kommunikationsprotokolle und ebenso bei der Übertragung von Komponentensystemen. Das vorgestellte Metamodell der Engineeringsprachen ist ein einheitliches Modell, wodurch ein einheitliches Verständnis der Komponentensysteme erreicht wird. Verwendet man das Metamodell zusammen mit einem vereinheitlichten Interface, können Engineeringsysteme mit kompatiblen Laufzeitumgebungen genutzt werden. Für die Entwicklung dieser Modelle sind neben dem zu modellierenden Gegenstand auch die zu modellierenden Facetten wichtig. Nur wenn diese insgesamt berücksichtigt werden, können die Modelle bei der Bewältigung der Herausforderungen helfen.

Die Frage, wie ein Nutzer von bestehenden für seine Aufgabenstellung relevanten Lösungen erfährt, ist in der Automatisierungstechnik ungelöst. Auf der Ebene der Komponenten findet eine Verteilung und eine anschließende Verwendung durch die Nutzer statt. Für Komponentensysteme gilt dies nicht. Ebenso finden eine Beschreibung und ein Monitoring der verwendeten Versionen von Komponenten nur implizit statt oder es wird gänzlich darauf verzichtet. Daraus resultieren Probleme bei der Wartung und Erweiterung von bestehenden Lösungen.

Die Merkmale sind nach dem Merkmalmodell zentral definiert. Die sich darauf beziehenden Systeme verwenden alle diese Definition, um die gleiche Semantik von Werten zu nutzen. In der Codeentwicklung werden Versionsverwaltungssysteme genutzt, damit bestehende Lösungen allen Entwicklern bekannt sind und Funktionalitäten nicht mehrfach entwickelt werden. Diese Mechanismen bilden eine Grundlage für das in dieser Arbeit vorgestellte Konzept.

5 Wiederverwendung in komponentenbasierten Architekturen

Ausgehend von den in Kapitel 3 beschriebenen Anforderungen und der Analyse der bisherigen Arbeiten (vgl. Kapitel 4) wird im Folgenden ein Konzept für die systematische Wiederverwendung von komponentenbasierten Lösungen vorgestellt. Durch das Konzept werden die drei Fragen aus Kapitel 1.2 beantwortet:

- Wie sieht ein Mechanismus für komponentenbasierte (Teil-)Lösungen aus, so dass sie wiederverwendet und auf andere Anwendungsfälle übertragen werden können?
- Wie wird der Wiederverwendungsmechanismus in bestehende Tools und Prozesse integriert, so dass der Nutzer ihn verwendet?
- Wie erfährt der Nutzer, dass eine geeignete Lösung existiert?

Die Antwort auf die erste Frage ist ein Mechanismus für die Wiederverwendung in Komponentensystemen. Die Delta-Modellierung hat sich in der Analyse des Stands der Technik als geeignet für die Anwendung in der Automatisierungstechnik erwiesen. Sie wird daher als Grundlage für den zu entwickelnden Wiederverwendungsmechanismus verwendet. In der späteren Nutzung müssen die Delta-Modelle gespeichert werden. Dies kann beispielsweise in Laufzeitumgebungen geschehen. Für diese Speicherung in Laufzeitumgebungen wird ein objektorientiertes Metamodell für Delta-Modelle vorgestellt.

Delta-Modelle transformieren ein System oder das Modell eines Systems in ein anderes System bzw. Modell. Wenn Delta-Modelle auf Modelle angewendet werden, müssen diese beschrieben sein. Das heißt, es müssen ein Metamodell des Modells und Anwendungsregeln vorliegen. Die Delta-Modelle erhalten durch das Metamodell des zu transformierenden Systems die Grundlage für ihre Anwendung (vgl. Abbildung 5.1). Wenn Elemente angelegt werden, müssen diese definiert und bekannt sein. In der Literatur werden Delta-Modelle ohne die Abstraktion durch eine Modellierungsebene auf Komponenten angewendet. Die Delta-Modelle könnten direkt auf die entsprechenden Komponentensysteme angewendet werden. Dagegen spricht allerdings, dass die Nutzbarkeit in diesem Fall auf das konkrete System beschränkt wäre. Entwickelte Komponentensysteme könnten nicht übertragen werden, da die Delta-Modelle auf anderen Systemen keine Anwendungsbasis hätten. Auch eine Anwendung der Delta-Modelle auf unterschiedliche Arten von Komponentensystemen (z. B. Funktionsbausteinnetzwerke oder P&I-Diagramme) wäre nicht möglich. Durch die heterogene Infrastruktur in der Automatisierungstechnik, d. h. durch die vielen verschiedenen Werkzeuge und Automatisierungssysteme, ist dieser Ansatz nicht zweckmäßig. Daher wird im folgenden Konzept ein generisches Komponenten-Metamodell als Basis für die Anwendung der Delta-Modelle entwickelt. Das Komponenten-Metamodell beschreibt hybride Systeme, d. h. Hard- und Softwarekomponenten. Es bildet eine Abstraktionsschicht von

den konkreten Komponenten und ermöglicht die unabhängige Handhabung der Komponentensysteme in Form von Modellen. So können die wichtigen Informationen, die in der Struktur und dem Zusammenschalten der Komponenten enthalten sind, wiederverwendet werden.

Eine Übersicht über die Zusammenhänge zwischen den Modellen ist in Abbildung 5.1 dargestellt. Das Komponenten-Modell setzt sich aus einem Typ-Modell und einem Instanz-Modell zusammen. In den Typ-Modellen sind die Typen der verwendeten Komponenten beschrieben. Zudem wird dort auf kompatible Komponententypen referenziert. So können mehrere kompatible Versionen von Komponententypen einem Typ-Modell zugeordnet werden. Die Zuordnung von unterschiedlichen kompatiblen Versionen zu einem Komponententyp wird durch die Referenz ermöglicht. Durch diese explizite Modellierung können diese kompatiblen Versionen aufgefunden und verwendet werden. Die konkrete Verschaltung der Komponenten wird in den darauf aufbauenden Instanz-Modellen modelliert. Im vorliegenden Konzept sind das Typ-, das Instanz- und das Delta-Metamodell enthalten. Diese beinhalten die Elemente und Vorschriften zur Bildung der zugehörigen Modelle. So beschreibt das Typ-Metamodell den Aufbau von Typ-Modellen.

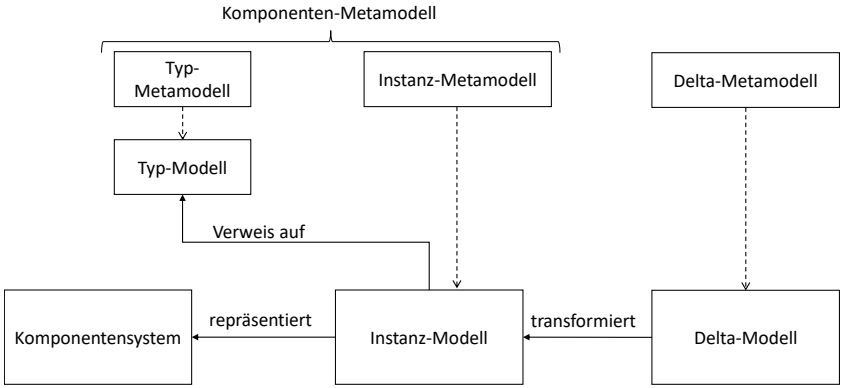


Abbildung 5.1: Übersicht der verwendeten Metamodelle, Modelle und deren Zusammenhänge.

Die zweite Frage fokussiert die Integration des Mechanismus zur Wiederverwendung in bestehende Prozesse und Werkzeuge. Sind in den bei der Entwicklung einer Automatisierungslösung beteiligten Werkzeugen die gleichen Typ-Modelle mit Referenzen auf entsprechende lokale Realisierungen vorhanden, können die Instanz-Modelle zwischen den Werkzeugen ausgetauscht werden. So können die durch Delta-Modelle beschrieben Lösungen in unterschiedlichen Anwendungsfällen zum Einsatz gebracht werden. Gleichzeitig verbessert dieser Ansatz die Interoperabilität zwischen den Systemen der Automatisierungstechnik.

Der angesprochene Interoperabilitätsaspekt ist zusätzlich Bestandteil der Antwort auf die zweite Frage. Damit die Nutzer das Konzept zur Wiederverwendung annehmen und verwenden, muss es sich in die bestehenden Prozesse und Werkzeuge einfügen. Das Konzept ist dafür additiv zu bestehenden Werkzeugen zu sehen und ist für den Nutzer im Regelbetrieb in der vollen Komplexität nicht sichtbar. Analog zu Versionsverwaltungssystemen

greift der Nutzer nur beim Austausch von Lösungen auf Schnittstellen des Konzepts zurück. Dies wird durch die Vorstellung von Prozessen für die Verwendung der Modelle unterstützt.

Für die Beantwortung der dritten Frage steht die Auffindbarkeit von Lösungen im Vordergrund. Der beste Wiederverwendungsmechanismus mit einer gelungenen Integration in Werkzeuge und Prozesse kann nicht genutzt werden, wenn der potentielle Nutzer nichts von existierenden zur Wiederverwendung geeigneten Lösungen weiß. Insbesondere gilt dies in der räumlich oder organisatorisch getrennten Entwicklung von Lösungen. Als Weg für die Auffindung existierender Lösungen ist ein zentraler Speicher für die Lösungen vorgesehen. Darin werden die gemeinsam verwendeten Typ-Modelle und die Delta-Modelle für die Beschreibung der Lösungen abgelegt. Die Nutzer können diesen Speicher durchsuchen und relevante Lösungen auf das lokale System herunterladen. Dort können diese verwendet und modifiziert werden.

Die angesprochenen Modelle, die für das Konzept benötigte Anwendungsumgebung und die Prozesse zur Wiederverwendung werden im Folgenden beschrieben. Das Kapitel ist folgendermaßen strukturiert: Abgeleitet von den Komponenten in der Automatisierungstechnik wird das Metamodell der Komponenten entwickelt (Abschnitt 5.1). Dieses ist die Grundlage für die Delta-Modellierung. Ausgehend von der Beschreibung von Variabilität durch Delta-Modelle in der Softwaretechnik wird das Metamodell eingeführt (Abschnitt 5.2). Dies wird an die Eigenschaften des Komponenten-Metamodells angepasst. In Abschnitt 5.3 wird die Verwendung der Modelle vorgestellt und anschließend auf die dezentrale Wiederverwendung eingegangen. Am Ende des Kapitels folgt eine kritische Diskussion des vorgestellten Konzepts.

5.1 Komponenten-Metamodell - Basis für die Wiederverwendung

Ausgehend von dem Modell für Engineeringsprachen, das ein Komponenten-Modell ist, wird ein Metamodell vorgestellt, das die Grundlage für die Deltamodellierung bietet.

Das Metamodell ist eine Erweiterung der in Abschnitt 2.2 vorgestellten Komponenten-Modelle. Diese wurden um Mechanismen des OPC UA Metamodells (vgl. Abschnitt 4.4) erweitert, sodass Referenzen zwischen Komponenten abgebildet werden können. Diese Referenzen bilden eine typisierte Beziehung zwischen Komponenten, die über die Übertragung von Informationen hinausgeht. So können beispielsweise logische Abhängigkeiten zwischen Komponenten modelliert werden.

5.1.1 Modellbeschreibung

Das Komponenten-Metamodell besteht aus einem Modellteil zur Beschreibung der Komponententypen und einem zweiten Teil, der die Komponentensysteme modelliert, die aus Instanzen dieser Komponententypen zusammengesetzt sind. Im Folgenden wird zunächst das Typ-Modell und anschließend das Instanz-Modell vorgestellt. Das Typ-Modell bildet die Basis, auf der das Delta-Modell ausgeführt wird. Alle verwendeten Komponententypen

werden durch dieses Modell beschrieben. Das Ergebnis der Anwendung des Delta-Modells ist ein Instanz-Modell. Dieses ist unabhängig von den konkreten Komponenten. Durch eine Transformation kann das Komponentensystem bestehend aus den realen Komponenten erzeugt werden.

Metamodell zur Beschreibung von Komponententypen

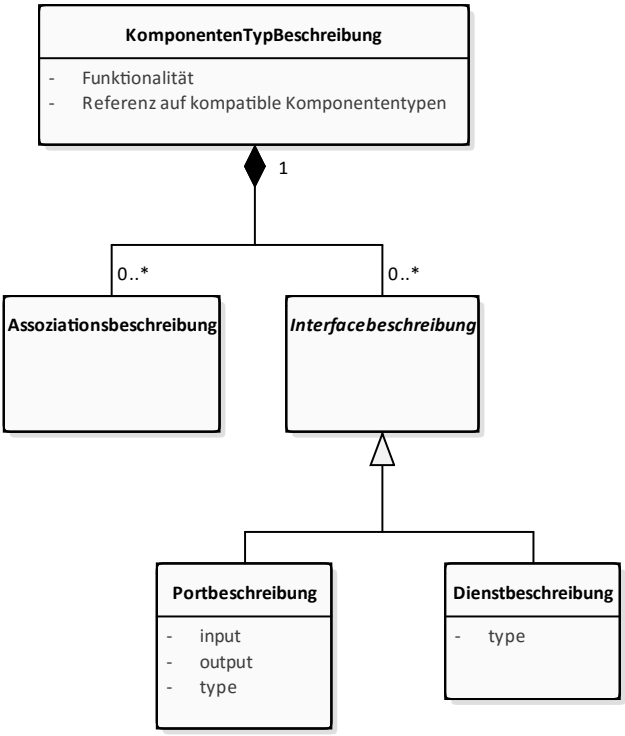


Abbildung 5.2: Metamodell der Komponententypen

Das Metamodell der Komponententypen ist in Abbildung 5.2 dargestellt. Dessen Kern ist die *KomponentenKlasse*, die Verweise auf mindestens eine Implementierung und eine Beschreibung der Funktionalität enthält. Die Klasse kann auf mehrere Versionen der Komponente verweisen. So können unterschiedliche kompatible Implementierungen explizit modelliert werden. Unterhalb der *KomponentenKlasse* wird das Interface durch die *Interfacebeschreibung* modelliert. Schnittstellenelemente sind sowohl Ports als auch Dienstschnittstellen. Diese Unterscheidung ist erforderlich, da Dienste eine andere Semantik im Hinblick auf die zugrunde liegende Verbindung haben. Beispielsweise werden Dienstnutzer

und Dienstempfänger möglicherweise erst zur Laufzeit festgelegt. So ist die Dienstschnittstelle sehr flexibel nutzbar.

Die Beschreibung der Funktionalität in der *KomponentenKlasse* dient dazu, den Zweck der konkreten Komponenten zu erkennen. Bei der Abbildung von Bausteinen wird über diesen Mechanismus festgehalten, welche Rolle der Funktionsbaustein ausfüllen kann. Für die Beschreibung der Funktionalität können verschieden komplexe Mechanismen zum Einsatz kommen. Die einfachste Umsetzung ist, die Funktionalität textuell, d. h. durch einen String, zu beschreiben. Der Vorteil ist, dass ein Mensch diesen sehr einfach interpretieren kann. Durch eine unzureichende Verwendung von Namenskonventionen kann es allerdings zu unterschiedlichen Beschreibungen der gleichen Funktionalität durch leicht unterschiedliche Namen kommen. Als Erweiterung kann eine zentrale Verwaltung dieser Bezeichnungen mit einer detaillierten Beschreibung realisiert werden. Analog zu der Definition von Merkmalen wird in einer Organisationseinheit oder darüber hinaus eine zentrale Grundlage geschaffen, wie die Funktionalität beschrieben werden kann und wie die Zusammenhänge zwischen diesen Funktionalitäten sind. Für diesen Ansatz existieren in der Forschung bereits Konzepte [Rie17] und es werden aktuell weitere Vorschläge zur Beschreibung von Fähigkeiten ausgearbeitet. In diesem Rahmen wird auch eine Beschreibung der Funktionalität mit formalen Methoden wie einer mathematisch-physikalischen Beschreibung oder mithilfe von Ablaufdiagrammen diskutiert. Diese weitergehenden Ansätze finden in der Praxis noch keine Verwendung. Im Wesentlichen mangelt es an konsistenten Definitionen der Funktionalität, die eine Bedeutung für ein größeres Anwendungsgebiet haben. Diese Ansätze sind für die vorliegende Arbeit zu weitgehend, können aber nachträglich ohne große Änderungen in das Gesamtkonzept übernommen werden. Für die weitere Betrachtung wird angenommen, dass es in der Organisationseinheit, in der das Konzept angewendet wird, eine konsistente Konvention zur Beschreibung der Funktionalität gibt, die von allen Anwendern benutzt wird.

Die Schnittstellen des modellierten Komponententyps beschreiben die Objekte der Klassen *Portbeschreibung* und *Dienstbeschreibung*. Durch die Portbeschreibung kann festgelegt werden, ob es sich um einen Ein- oder Ausgang handelt. Zusätzlich kann der Typ des Ports beschrieben werden. Durch die Dienstbeschreibung wird ein vorhandenes Dienstinterface abgebildet. In ihr ist hinterlegt, ob es sich um einen Dienstaufruf oder das Anbieten eines Dienstes handelt. Zusätzlich werden die relevanten Dienste beschrieben.

Metamodell zur Modellierung von Komponentensystemen (Instanz-Modell)

Mit Objekten der Klassen des Instanz-Modells werden konkrete Systeme modelliert. Das Modell ist in Abbildung 5.3 dargestellt. Kern des Modells ist das *Komponenten-Systemmodell*, das aus Objekten der Klassen *KomponentenInstanz* und *Schnittstellenelement* besteht. Diese bilden das Interface des *Komponenten-Systemmodells* nach außen und dessen internen Aufbau ab. Ein Schnittstellenelement kann durch Dienstschnittstellen und Ports realisiert werden. Da ein Komponenten-Systemmodell nach außen die Schnittstellen einer Komponente bereitstellt, können die Beschreibungsobjekte in beiden Fällen verwendet werden. Mit diesem Modell wird ausgehend von den *KomponentenInstanzen* der Aufbau eines konkreten Komponentensystems beschrieben. Es ist zu erkennen, dass die *KomponentenInstanz* über Ports als Ein- und Ausgänge verfügt. Zwischen den Ports können gerichtete

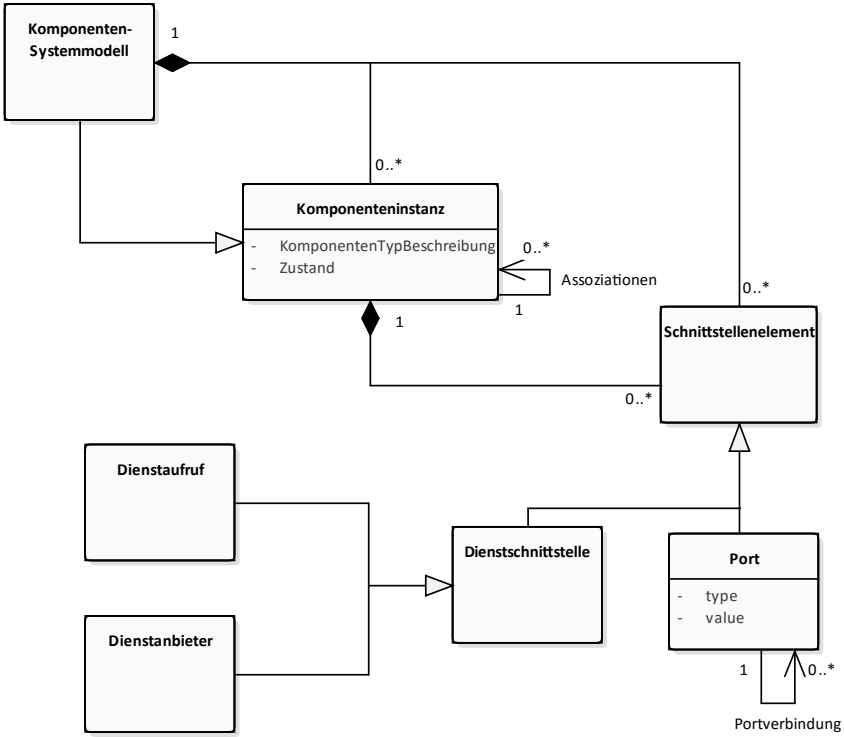


Abbildung 5.3: Metamodell zur Abbildung von Systemen bestehend aus Komponenten.

Verbindungen aufgebaut werden. Die Semantik der Verbindung besteht darin, dass Informationen, Stoffe oder Energie vom Anfang zum Ziel der Verbindung transportiert wird. Je nach Typ des Ports kann es sein, dass beispielsweise ein Massenfluss oder ein Informationsfluss modelliert wird. Abhängigkeiten zwischen Komponenten, die nicht mit einer Verbindung über einen der Ports in Zusammenhang stehen, können durch Assoziationen dargestellt werden.

Durch die Klassen *Dienstaufruf* und *Dienstanbieter* wird die Einbindung eines Komponentensystems in eine Dienstarchitektur abgebildet. Die Komponenten des Systems können sowohl Diensterbringer als auch Dienstnutzer sein. Für die Ausgestaltung einer Dienstschnittstelle existieren unterschiedliche Herangehensweisen. Exemplarisch sind das Dienstmodell [DIN14] und die Integration von Diensten [BFK+17, WE17] in die Automatisierungstechnik. Es ist allgemeiner Konsens, dass eine solche Anbindung in Zukunft benötigt wird. Damit das Modell in diesem Kontext anwendbar ist, ist die Modellierung dieser Dienstschnittstelle vorgesehen.

Das Instanz-Modell ist analog zum in [WGE+18] vorgestellten Modell für die Sprachen des Engineerings. Die Beschreibung von geschachtelten Systemen ist in diesem Modell in ähnlicher Weise gelöst worden. Die *Komplexen Typen* setzen sich aus Basis Typen und Schnittstellenkomponenten zusammen. Das Modell sieht zusätzlich die Möglichkeit vor, Verbindungen zwischen Komponenten als Objekte zu modellieren. Dies ist erforderlich, um für die Engineering-Systeme einen einheitlichen Aufbau zu erreichen und die Verbindungen als eigene Entitäten verwalten zu können. In den Laufzeitsystemen kann es sinnvoll sein, die Verbindungen als eigene Objekte zu realisieren. So können beispielsweise Transformationen von Datentypen umgesetzt werden.

5.1.2 Anwendungsregeln für die Komponenten-Metamodelle

Neben den beschriebenen Modellzusammenhängen sind für die Nutzung der Modelle Anwendungsregeln erforderlich. Diese spezifizieren die Verwendung der Modellelemente und legen einen Rahmen für deren Zusammensetzung fest.

Regel 1: Während der Transformation eines Modells kann dieses von den Regeln abweichen.

Regel 2: Verbindungen müssen immer von einem Port an der Quelle zu einem oder mehr Ports an der Senke führen.

Regel 3: Komponenten und Verbindungen können nur innerhalb eines Komponentensystems verortet sein. Wird ein System gelöscht, werden alle enthaltenen Elemente ebenfalls entfernt.

Regel 4: Für die Korrektheit der Verbindungen gelten die jeweiligen Regeln des modellierten Systems.

Regel 5: Regeln des modellierten Systems, die Abhängigkeiten zwischen Komponenten betreffen, werden über Assoziationen abgebildet.

Regel 6: Alle in einem Instanz-Modell verwendeten Komponenten müssen auf eine Komponente in einem Typ-Modell referenzieren.

Regel 7: Das Interface einer Komponente im Instanz-Modell muss dem Interface des referenzierten Komponententyps entsprechen.

Regel 8: Über die Systemgrenzen hinweg können Informationen nur über die Ports und Dienstschnittstellen des Systems übertragen werden.

In Regel 1 ist der Geltungsbereich der Regeln auf nicht in einer Transformation befindliche Modelle eingeschränkt. Wenn ein Modell verändert wird, kann temporär von den Vorgaben der Regeln abgewichen werden. Nach Abschluss der Transformation muss ein regelkonformes Modell vorliegen.

Verbindungen müssen nach Regel 2 immer Quelle und Senke verbinden. Eine Verbindung ohne Anfangs- und/oder Endpunkt darf es nicht geben. Dies entspricht der grundsätzlichen Wahrnehmung von Verbindungen, die eine reine Übertragungsfunktion haben. Wenn ein Rohr ohne Abschluss verbliebe, würde alles, was hineinfließt, unkontrolliert herausfließen. Bei Softwaresystemen können einseitige Verbindungen bestehen, welche möglicherweise

allerdings auf einen Fehler hindeuten. Im Rahmen einer Transformation ist es temporär zulässig, Verbindungen unvollständig zu belassen.

Komponenten und Verbindungen dürfen nur innerhalb eines Instanz-Modells angelegt werden (Regel 3). Da eine Komponente ein Komponentensystem sein kann, entsteht ein hierarchisches System. Jede Komponente und Verbindung muss darin eindeutig verortet sein. Es muss diskutiert werden, was passiert, wenn eine zusammengesetzte Komponente gelöscht wird. Die einfachste Möglichkeit ist die Löschung aller in ihr enthaltenen Komponenten und Verbindungen. Die Kompositionsbeziehung zwischen dem System und seinen Bestandteilen unterstützt dieses Vorgehen. Allerdings ist es auch denkbar, dass nur die Systemhülle entfernt wird und die enthaltenen Elemente Teile des darüber liegenden Systems werden. Dieses Vorgehen hat den Nachteil, dass das Löschen von Systemen kontextsensitiv wird. Handelt es sich bei dem zu löschenden System um ein Subsystem, bleiben die Elemente erhalten. Ist es kein Subsystem, müssen die Elemente gelöscht werden. Daher wird in Satz 2 von Regel 3 festgelegt, dass Elemente eines Systems bei dessen Löschung ebenfalls gelöscht werden.

Das modellierte System legt Regeln fest, welche Typen von Ports miteinander verbunden werden können (Regel 4). Das Metamodell ermöglicht die Verbindung von allen Ports mit allen anderen Ports. Ob diese Verbindungen zulässig sind, ergibt sich aus den Konvertierungsregeln des Systems. Nach diesen Regeln kann das Instanz-Modell nach der Erstellung geprüft werden. Abhängigkeiten zwischen Komponenten des modellierten Systems, die über Verbindungen hinausgehen, werden nach Regel 5 über die Assoziationen modelliert. Beispielsweise kann eine Vorgänger-Nachfolger Beziehung auf diese Weise abgebildet werden.

In einem Instanz-Modell verwendete Komponenten müssen auf eine Komponente in einem Typ-Modell referenzieren (Regel 6). Typ-Modelle definieren die Komponententypen, die in Instanz-Modellen verwendet werden können. Wenn die Referenz nicht vorhanden ist bzw. wenn kein passendes Typ-Modell existiert, ist die Komponente nicht definiert und kann daher nicht verwendet werden. In diesem Kontext ist Regel 7 zu verstehen. Die Komponente im Instanz-Modell muss das gleiche Interface aufweisen wie die Definition ihres Typs. Weichen die Modelle voneinander ab, liegt eine Inkonsistenz vor, die dokumentiert und behoben werden muss. In Regel 8 wird die Vorgabe aus [IEC04] aufgegriffen, dass Komponenten abgeschlossen sein müssen. Diese wird auf die Komponentensysteme ausgedehnt und die Übertragung von Informationen auf die explizit modellierten Schnittstellen begrenzt.

Mit den Regeln werden Einschränkungen hinsichtlich des Verhaltens und Aufbaus der abgebildeten Systeme formuliert. Beispielsweise kann ein System, das keinen strukturierten Aufbau besitzt, nicht von dem vorgestellten Komponenten-Metamodell abgebildet werden. Zusätzlich wird die Abhängigkeit des Modells von den Regeln des modellierten Systems verdeutlicht. So kann das Metamodell keine Aussage über eine etwaige Kompatibilität von Typen von Ports treffen.

Die vorgestellten Regeln definieren Modellzustände, die als Basis für die weitere Nutzung dienen. Ausgehend von einem konformen Modell kann die anschließende Bearbeitung der Modelle erfolgen.

5.1.3 Einordnung des Komponenten-Metamodells

Das Komponenten-Metamodell abstrahiert sowohl von den Komponententypen als auch von den aus Komponenteninstanzen gebildeten Netzwerken. Es werden jeweils nur das Interface sowie die auszuführende Funktionalität modelliert. Aus einer abstrakten Sicht ist das vorgestellte Modell ein Rollenmodell (vgl. Kapitel 2.2.1). Durch die Trennung von Implementierung und der geforderten Funktionalität sind die entwickelten Komponentenstrukturen in unterschiedliche Implementierungen überführbar. Im Kontext der Bausteinsprachen und Softwarekomponenten wird in [Ens01] ein Ansatz für die implementierungsunabhängige Beschreibung von Bausteinnetzwerken vorgestellt. Neben der Fokussierung auf Softwarekomponenten ist das Modell für den in dieser Arbeit vorgesehenen Anwendungsfall nicht gänzlich geeignet. Es werden beispielsweise ausschließlich Signalverbindungen zwischen Komponentenports modelliert und die Typen der Ports sind auf die in der Softwaretechnik gebräuchlichen beschränkt.

Ein reines Rollenmodell legt den Schwerpunkt auf die Beschreibung der geforderten Funktionalität und additiver Anforderungen (z. B. durch Merkmalausprägungsaussagen). Im Gegensatz dazu werden im vorgestellten Modell zudem die Mindestanforderungen an das Interface der repräsentierten Komponenten festgelegt.

Das Modell vereint Elemente von ECL- und ER-Modell (vgl. Abschnitt 4.4.3). Es können Informations-, Energie- oder Materieflüsse zwischen den Systemelementen modelliert werden (ECL-Modell). Durch die Assoziationen sind logische Abhängigkeiten zwischen Systemelementen abbildbar (ER-Modell). Die vorgenommene Zusammenführung der zwei Paradigmen in einem Modell erlaubt die Abbildung einer großen Bandbreite von Systemen. Eine Reduktion des Modells um eines der Paradigmen führt entweder zu einer geringeren Nutzbarkeit des Modells oder die jeweils fehlende Modellierungsmöglichkeiten werden mit den existierenden Modellelementen umgesetzt. Durch die Verbindung der zwei Paradigmen entsteht ein einheitliches und durchgängiges Modell, das flexibel anwendbar ist. Beispiele für abzubildende Systeme werden in Abschnitt 5.1.4 dargestellt.

Auf die explizite Modellierung der Ausführungssemantik der Komponenten wird im vorliegenden Modell verzichtet. Insbesondere bei Systemen bestehend aus Hardwarekomponenten ist keine Ausführungssemantik nötig. Im Kontext von Steuerungen kann z. B. eine Reihenfolge der Ausführung durch Assoziationen zwischen den Komponenten abgebildet werden. Alternativ kann die Ausführungsreihenfolge auch durch eine Heuristik im Laufzeit- bzw. im Engineeringsystem festgelegt werden.

5.1.4 Abgebildete Implementierungen

Im folgenden Abschnitt werden Beispiele für die Verwendung des Komponenten-Metamodells vorgestellt. Dabei wird auf die Erstellung solcher Modelle eingegangen.

Mit der *KomponentenInstanz* des vorgestellten Modells können verschiedene Arten von Komponenten abgebildet werden. Es kann sich dabei um Soft- oder Hardwarekomponenten handeln. Nach der Definition der technischen Komponente (vgl. Abschnitt 2.2) können diese gleich behandelt werden. In der Automatisierungstechnik liegt der Kontext der betrachteten Systeme auf beiden Arten von Komponenten. Durch die engen Verflechtungen

und Abhängigkeiten ist es erforderlich, Soft- und Hardware gleichermaßen im Modell eines Systems zu berücksichtigen. So kann der Zusammenhang zwischen einem Aktor und der zugehörigen Steuerung abgebildet werden. Sollte beispielsweise in einer Package Unit der Motor getauscht werden, kann der Informationsfluss zu der entsprechenden Steuerungskomponente dargestellt werden. Die Genauigkeit, mit der das Modell des konkreten Systems entwickelt wird, hängt von den jeweiligen Anwendungsfällen ab. Im Hinblick auf die Package Units sind verschiedene Grade der Modellierung denkbar. So könnte die Kommunikation zwischen dem Aktor und der Steuerung ebenso als Komponente modelliert werden. Eine detailliertere Darstellung kann durch die additive Modellierung von dazwischen liegenden Hardwarekomponenten erreicht werden. Wenn diese im Fokus der Betrachtung sind, müssen sie in dem Modell berücksichtigt werden.

Es ist zu beachten, dass die Semantik der Verbindungen zwischen den Komponentenports nicht gleich der von Signalverbindungen oder Informationsverbindungen aus der Softwaretechnik ist. Sie ist eine Kombination aus der Übertragung von Informationen bzw. Signalen und dem Transport von Stoffen oder Energie. Die Verbindungen verknüpfen aus Sicht des Modells nur zwei Ports von Komponenten. Was übertragen wird, ergibt sich aus dem Typ der Ports. Das Modell erlaubt die Verbindung von zwei Ports unterschiedlichen Typs. Die Prüfung, ob dieses Vorgehen sinnvoll ist und ob eine geeignete Konvertierungsvorschrift (vgl. Konvertierung von Datentypen nach [IEC14b]) vorliegt, muss durch den Anwender bzw. durch die Implementierung erfolgen.

In Abbildung 5.4 ist beispielhaft das Modell eines Systems bestehend aus zwei Komponenten dargestellt. Auf der Systemgrenze sind die Schnittstellen des Systems nach außen zu erkennen. Die beiden Eingangsports des Systems sind mit den Eingangsports der linken Komponente verknüpft. Deren Ausgang ist mit dem Eingang der rechten Komponente verbunden. Die beiden Ausgänge der rechten Komponente werden über die Ausgangsports des Systems von außen zugänglich. Der gestrichelte Pfeil stellt eine Assoziation dar. In diesem Fall wird die Ausführungsreihenfolge von Softwarekomponenten abgebildet.

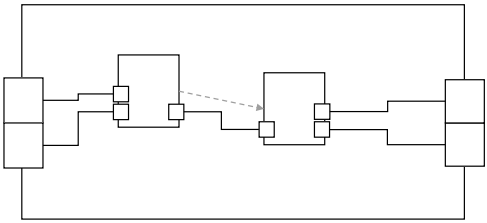


Abbildung 5.4: Beispiel eines Systemmodells mit zwei Komponenten.

Ein Beispiel für das Modell eines hybriden Systems ist in Abbildung 5.5 dargestellt. Im unteren Teil der Abbildung ist ein System aus zwei Pumpen und einem Tank abgebildet. Im oberen Teil ist die zugehörige Steuerung beispielhaft modelliert. Die Steuerung ist ebenso als Komponentensystem modelliert und stellt ein Subsystem im Gesamtsystem dar. Dies ist ein Beispiel für ein hierarchisch aufgebautes Systemmodell. Die weiteren Komponen-

ten könnten ebenso wiederum Komponentensysteme sein. Für die Darstellung ist es nicht sinnvoll, alle Komponentensystem mit ihrem inneren Aufbau zu zeigen. Zur Reduktion der Komplexität kann der innere Aufbau verborgen und das Komponentensystem als Blackbox gezeigt werden. In der Abbildung ist zu erkennen, dass die Pumpen und der Tank jeweils Schnittstellen zur Steuerung besitzen. Den Pumpen werden Sollwerte für die Drehzahl vorgegeben. Aus einem Füllstandsensor des Tanks werden Informationen über den Zustand des Systems für die Steuerung gewonnen. Der Detailgrad des Modells muss spezifisch für jeden Anwendungsfall gewählt werden.

In welcher Form die Beschreibung der Komponente vorliegt, ist für die Nutzung des Modells unerheblich. Bei der Modellierung von Hardwarekomponenten kann die Grundlage eine Typenbeschreibung in Form eines AutomationML Modells sein. Analog dazu kann auf der Softwareseite die Klasseninformation oder der Prototyp einer Komponente verwendet werden. Eine bestehende (Teil-)Lösung kann modelliert werden und steht so für die Wiederverwendung in anderen Bereichen zur Verfügung. Für die Bildung eines Modells werden die typisierten Ein- und Ausgänge der Komponente sowie eine Beschreibung ihres Verhaltens benötigt. Darüber hinaus ist ein Verweis auf die Realisierung der Komponente erforderlich. Dies kann bei einer Hardwarekomponente eine Typen-ID (z. B. Bestellnummer) sein. Im Bereich der Software ist die Angabe von Klassennamen oder der Verweis auf einen Prototyp sinnvoll.

Je nachdem, wie die Komponente bzw. ihre Beschreibung vorliegt, variieren die Möglichkeiten, das korrespondierende Modell zu erzeugen. Am wenigsten Anforderungen an die Beschreibung der Komponente stellt die manuelle Erzeugung des Modells. Hierbei ist es unerheblich, ob der durchführende Mensch das Modell anhand der Betrachtung eines physischen Gegenstandes oder einer Typinformation aufbaut. Die Komponente muss nicht beschrieben sein. Gleichwohl ist es sinnvoll, die Art der Interpretation zu formalisieren (z. B. durch die Vorgabe von Richtlinien), damit die Modelle ähnlich aufgebaut sind. Alternativ kann das Modell automatisiert aus einer Komponentenbeschreibung, d. h. aus einer Typ- oder Instanzbeschreibung, generiert werden. Grundlage dafür ist eine Beschreibung des Interfaces der Komponente. Möglich ist beispielsweise die Auswertung von Datenblättern oder die Interpretation eines AutomationML-Modells. Die Auswertung einer Beschreibung ist sowohl für Hard- als auch für Softwarekomponenten umsetzbar. Die dritte Möglichkeit ist die automatisierte Identifizierung der Komponente selbst. Dies könnte beispielsweise durch die Erkundung einer Komponente auf einem OPC UA Server erfolgen. Durch die vorliegende Typisierung der Objekte kann das Modell der Komponente erzeugt werden. Für Softwarekomponenten ist diese Erzeugung bei sich selbst beschreibenden Laufzeitsystemen gut möglich. Dies ist bei der automatisierten Identifizierung von Hardwarekomponenten nicht der Fall. Ohne Zuhilfenahme von Typinformationen ist die direkte Erzeugung des Modells allein aus der vorliegenden Hardwarekomponente nur mit einem großen Aufwand im Hinblick auf die verwendeten Sensoren und die Fusion der Sensorwerte möglich. Zusammenfassend lässt sich festhalten, dass eine Automatisierung der Erzeugung durch eine intrinsische Erkundbarkeit der Komponenten oder der Typbeschreibung vereinfacht bzw. überhaupt erst ermöglicht wird.

Auf der Typenebene des Modells wird bei der Erstellung bewusst auf eine Modellierung von Vererbungsbeziehungen zwischen Komponenten-Instanzen verzichtet, da diese nicht Modellierungsgegenstand sind. Ebenso wird auf eine Modellierung von zusammengesetzten

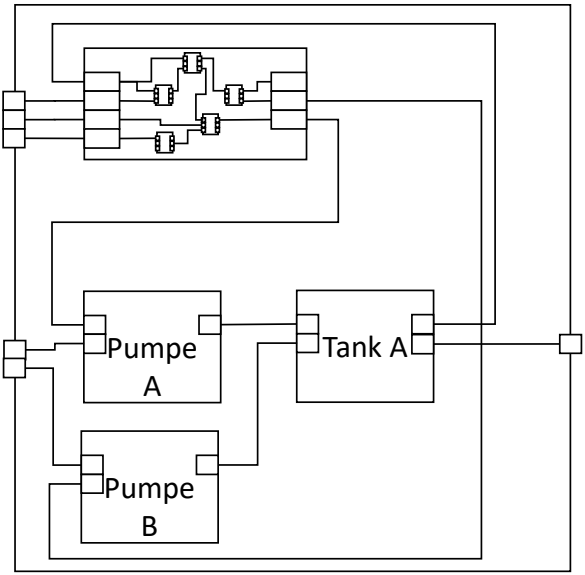


Abbildung 5.5: Beispiel für das Modell eines hybriden Instanz-Modells

Komponenten verzichtet. Diese werden über den Verweis auf die entsprechende Implementierung berücksichtigt, d. h., wenn eine zusammengesetzte Komponente modelliert werden soll, wird sie in der Außensicht als Komponente beschrieben. Ein analoges Vorgehen findet auf der Typenebene statt. Komponenteninstanzen repräsentieren auch zusammengesetzte Komponenten und sind somit wieder selbst Komponentensysteme.

Systeme, die nicht aus Komponenten aufgebaut sind, können nicht durch das Metamodell abgebildet werden. Beispiele sind unstrukturierte Systeme bzw. Systeme, über deren internen Aufbau keine Informationen vorliegen.

5.2 Δ – Metamodell

Im folgenden Abschnitt wird das Delta-Metamodell vorgestellt. Ausgangspunkt des Modells ist die vorgestellte Systematik der Modellierung von Variabilität im Lösungsraum durch die Nutzung von Transformationen (vgl. Abschnitt 4.3.3). Für die Nutzung der Delta-Modelle innerhalb der zunehmend mehr verwendeten Laufzeitumgebungen und im Hinblick auf die flexiblen und wandelbaren Produktionssysteme der Zukunft ist eine explizite Modellierung der Deltas als Objekte in einer Laufzeitumgebung ein bewährter Ansatz. Die Abbildung von Strukturen in objektorientierten Laufzeitumgebungen wurde bereits bei anderen deskriptiven Modellen verwendet (vgl. Merkmale und Merkmal-Ausprägungsaussagen,

Abschnitt 4.4.3). So können die modellierten Deltas in einem Laufzeitsystem abgelegt und jederzeit durch autorisierte Systeme und Nutzer erkundet werden. Nachfolgend wird zunächst das Modell vorgestellt.

5.2.1 Modellbeschreibung

Ein Delta-Modell beschreibt die Transformation von einem Komponenten-Systemmodell in ein anders. Die Transformation ist in Gleichung 5.1 dargestellt. Das $KomponentenSystemmodell_n$ wird durch Anwendung des $DeltaModell_i$ in $KomponentenSystemmodell_{n+1}$ transformiert. Das Systemmodell, das die Ausgangsbasis darstellt, muss den in Abschnitt 5.1.2 vorgestellten Regeln entsprechen. Dies gilt ebenso für den Zielzustand der Transformation. Während der Transformation kann von den vorgestellten Regeln abgewichen werden.

$$KomponentenSystemmodell_{n+1} = KomponentenSystemmodell_n + DeltaModell_i \quad (5.1)$$

Das Delta-Modell ist eine Zusammenfassung von Transitions-Operationen. Die in einem Delta-Modell enthaltenen Operationen transformieren ein Instanz-Modell von einem konsistenten in einen anderen konsistenten Zustand. Aus der jeweiligen Zusammenstellung der Operationen in einem Delta-Modell und der Anforderung ein konsistentes Ergebnis der Transformation zu erreichen, ergeben sich Randbedingungen für das Instanz-Modell, das transformiert werden soll. Die Operationen müssen zu dem Kontext, in dem sie angewendet werden, passen. Ist dies nicht der Fall, kann das Ergebnis ein inkonsistentes Instanz-Modell sein.

In Abbildung 5.6 ist das Delta-Metamodell als UML-Modell dargestellt. Kern des Modells ist das Delta-Objekt, das über einen Namen eindeutig identifizierbar ist. Ein Delta-Objekt besteht aus einer sortierten Menge von Operationen. Diese Operationen sind spezielle Ausprägungen der Grundoperationen *Hinzufügen*, *Löschen* und *Modifizieren* der Delta-Modelle. Für das vorgestellte Komponenten-Metamodell wurden die Operationen stärker ausdifferenziert. Es sind die Operationen für das Hinzufügen und Löschen von Objekten (Komponenten und Schnittstellenelementen) sowie für das Hinzufügen und Löschen von Assoziationen und Verbindungen vorgesehen. Zusätzlich können der interne Zustand einer Komponente oder der Wert eines Parameters über die entsprechenden Zustände geändert werden. Die Ausdifferenzierung der Operationen in die unterschiedlichen Spezialformen kann auf den ersten Blick zu feingranular wirken und führt in der Anwendung zu komplexeren Modellen. Allerdings wird durch die Ausdifferenzierung klar erkenntlich, ob es sich bei der vorgenommenen Transformation um eine Änderung der Schnittstelle des Instanz-Modells handelt oder „nur“ der interne Aufbau variiert wird. Diese semantische Trennung ist ähnlich zur Unterscheidung von externer und interner Variabilität. Ebenso unterschiedlich sind das Setzen von Parametern und die Veränderung eines internen Zustands. Die Operationen werden in einer konkreten Reihenfolge angeordnet. Dies ist in der Abbildung als verkettete Liste von Elementen dargestellt.

In der Literatur ist eine Reihenfolge der Delta-Operationen nicht vorgesehen, da diese durch eine geeignete Reihung der Typen von Operatoren (Addition, Subtraktion, Modifikation)

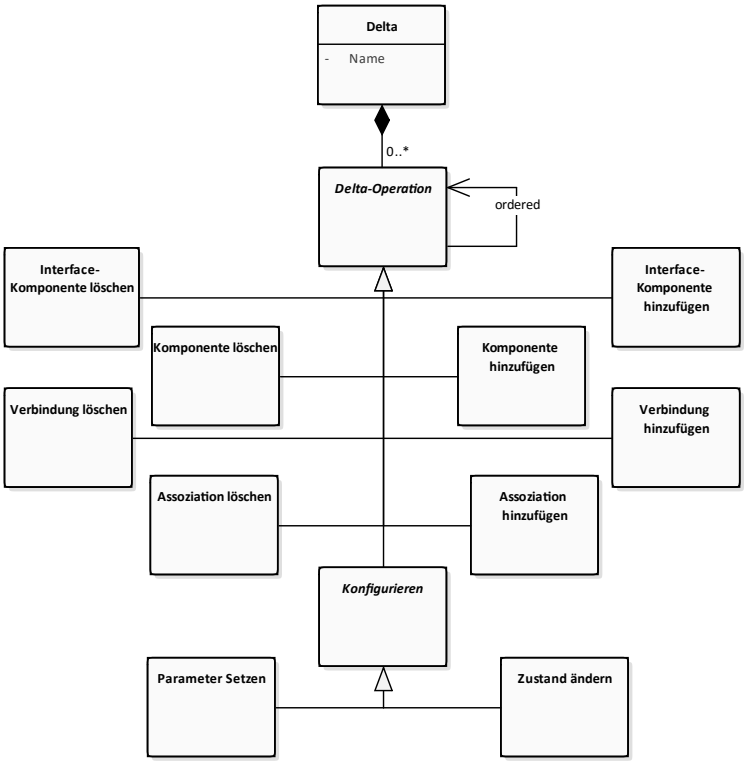


Abbildung 5.6: Übersicht über das Delta-Metamodell.

nicht nötig ist. Im Kontext von Softwaresystemen ist der Verzicht auf eine Anwendungsreihenfolge schlüssig und richtig, insbesondere wenn das zugrundeliegende Komponentensystem die Schachtelungstiefe eins besitzt. In diesem Fall gibt es keine Abhängigkeit zwischen Operationen, die das Anlegen von Komponenten durchführen. Wenn im Komponentensystem eine Hierarchie existiert, so ist es plausibel, erst das Elternobjekt anzulegen und anschließend die Kinderobjekte. Im allgemeinen Fall kann davon ausgegangen werden, dass erst das Elternobjekt angelegt werden muss, bevor das Kindobjekt darunter angelegt werden kann. Wenn beispielsweise ein Tank nicht existiert, kann er nicht befüllt werden. Für diese Fälle ist die Modellierung einer Reihenfolge in den Delta-Operationen erforderlich. Dies kann entweder über eine die Reihenfolge abbildende containment-Beziehung zwischen den Operationen und dem entsprechenden Delta oder über eine Modellierung im Delta-Metamodell erfolgen. Die Modellierung im Delta-Metamodell kann beispielsweise durch die Nutzung von Parametern in den Operationen oder von Vorgänger-/Nachfolger-Referenzen zwischen den Operationen umgesetzt werden. Bei der Anwendung der Delta-Operationen ist die Reihenfolge entsprechend zu beachten.

Der Nutzen des Delta-Modells für die Beschreibung der Variabilität soll exemplarisch an einem Beispiel verdeutlicht werden. Ausgangsbasis ist das Systemmodells aus zwei Komponenten aus Abbildung 5.4. Dieses Modell wird mittels folgendem Delta-Modell in das Systemmodell aus drei Komponenten transformiert.

Das Delta-Modell besteht aus folgenden Operationen:

1. **Verbindung löschen:** Löschen der Verbindung der rechten Komponenten zum Ausgangsport des Systems.
2. **Interface-Komponente hinzufügen:** Hinzufügen des dritten Eingangsports des Systems.
3. **Komponente hinzufügen:** Hinzufügen der dritten Komponente.
4. Dreimal **Verbindung hinzufügen:** Hinzufügen der Verbindungen vom Eingangsport zur dritten Komponente, von der rechten Komponente zur dritten Komponente und von der dritten Komponente zum Ausgangsport.
5. **Assoziation hinzufügen:** Assoziation von der rechten Komponente zur dritten Komponente hinzufügen.

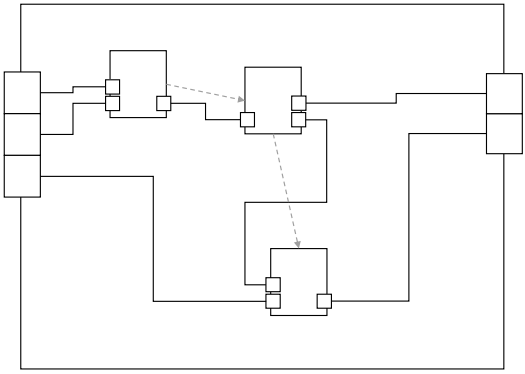


Abbildung 5.7: Beispiel eines Systemmodells mit zwei Komponenten.

Dieses einfache Beispiel verdeutlicht den intuitiven Charakter des Ansatzes. Zusätzlich wird klar, dass die Anzahl der Operationen in einem Delta-Modell mit der Anzahl der Änderungen linear wächst. Das Delta-Modell kann für die Transformation von jedem Systemmodell in jedes beliebige andere Systemmodell genutzt werden. Mit der Betrachtung dieses Sachverhalts beschäftigt sich Abschnitt 5.2.2.

5.2.2 Variantenbeschreibung mit Delta-Modellen

Das vorgestellte Modell zur Deltamodellierung ermöglicht es, Varianten von Komponentensystemen im Lösungsraum zu beschreiben, d. h., Variationen der Implementierung können

modelliert werden. Zusätzlich zu der in Abschnitt 4.3.1 vorgestellten Definition ist eine Variante im Kontext der vorgestellten Delta-Modellierung das Produkt, das nach Anwendung eines Delta-Modells auf ein Instanz-Modell entsteht. Delta-Modelle können durch die zugrundeliegenden Operationen tiefgreifende Veränderungen an Systemen vornehmen. Im Extremfall kann dies der vollständige Austausch des Ausgangssystems sein. Es ist möglich, zwei Komponentensysteme als Varianten voneinander zu modellieren, die wenige oder gar keine Gemeinsamkeiten besitzen. Im angesprochenen Extremfall würde das erste Komponentensystem komplett gelöscht und anschließend komplett aufgebaut werden. Diese massiven Veränderung widersprechen dem Gedanken der weitgehenden Wiederverwendung und sind eher der regelbasierten Entwicklung zuzuordnen. Es bedarf daher einer Unterteilung bzw. einer Begrenzung der Delta-Modelle, um zusammengehörige Systeme identifizieren zu können.

Eine Möglichkeit, die Größe der Deltas zu begrenzen, ist die Einführung eines Abstandsmaßes und die Definition einer zulässigen Obergrenze. Im Kontext der Variabilität existieren verschieden Abstandsmaße. Allerdings kann nur subjektiv entschieden werden, ob ein System Variante eines anderen ist. Für die Delta-Modelle bieten sich ein an das Euklidsche Abstandsmaß angelehntes Maß an. Dafür wird die gewichtete Summe der Anzahl der einzelnen Operationen in einem Delta gebildet.

$$S = S_{ExInt} + S_{Kom} + S_{Ver} + S_{Asso} + a_9 \cdot OP_{mod} \tag{5.2}$$

In Gleichung 5.2 ist die Berechnungsvorschrift dargestellt. Sie besteht aus den gewichteten Einzelsummen für die jeweiligen Operationen, die in den Gleichungen 5.3 bis 5.6 angegeben sind. In Gleichung 5.3 werden die Operationen, die das externe Interface des Komponentensystems verändern, berücksichtigt. Mit dem Faktor a_1 wird die Anzahl der Operationen, die dem externen Interface einen Bestandteil hinzufügen, gewichtet. a_2 ist der Gewichtungsfaktor für die Anzahl der Operationen, die das externe Interface verkürzen. Analog werden die gewichteten Summen für die internen Komponenten S_{Kom}, die Verbindungen S_{Ver} und Assoziationen S_{Asso} gebildet.

$$S_{ExInt} = a_1 \cdot ExInt_{add} + a_2 \cdot ExInt_{del} \tag{5.3}$$

$$S_{Kom} = a_3 \cdot Kom_{add} + a_4 \cdot Kom_{del} \tag{5.4}$$

$$S_{Ver} = a_5 \cdot Ver_{add} + a_6 \cdot Ver_{del} \tag{5.5}$$

$$S_{Asso} = a_7 \cdot Asso_{add} + a_8 \cdot Asso_{del} \tag{5.6}$$

Die Parameter müssen spezifisch für den jeweiligen Anwendungsfall festgelegt werden. Es kann beispielsweise sinnvoll sein, die Änderung des externen Interfaces stärker zu gewichten als eine Manipulation des internen Aufbaus. Alternativ könnten auch alle Faktoren auf den Wert eins gesetzt werden, sodass nur die reine Anzahl der Operationen unabhängig von der Wirkungsweise berücksichtigt wird. In der praktischen Anwendung ist es sinnvoll, die komplementären Operationen (Hinzufügen und Löschen) mit dem gleichen Faktor zu gewichten. Erfolgt dies nicht, führt beispielsweise das Löschen von Verbindungen zu einem größeren Abstandsmaß als das Hinzufügen.

Die in Gleichung 5.2 angegebene gewichtete Summe (S) stellt ein absolutes Abstandsmaß dar. Es wird ausgehend von der Anzahl der Operationen eine Berechnungsmethode für die Unterschiedlichkeit von Delta-Modellen angeboten. Allerdings wird bei dieser Art der Berechnung nicht berücksichtigt, wie groß der Anteil der Änderungen an dem Ausgangssystem ist. So können drei Operationen an einem kleinen Ausgangssystem eine größere Veränderung darstellen als drei Operationen an einem großen Modell. Um diesen Umstand zu berücksichtigen, ist es sinnvoll, ein auf die Größe des Ausgangssystems normiertes Abstandsmaß zu berechnen. Für die Normierung der Summe bieten sich sowohl die reine als auch eine gewichtete Anzahl der Systembestandteile an. Durch hohe Gewichtungsfaktoren bei der Berechnung von S kann es zu einer starken Verzerrung kommen. Um dies zu kompensieren, müssen die Systembestandteile mit entsprechenden Faktoren gewichtet werden.

$$S_N = \frac{S - a_9 \cdot OP_{mod}}{\frac{|a_1|+|a_2|}{2} \cdot ExInt_{Anz} + \frac{|a_3|+|a_4|}{2} \cdot Kom_{Anz} + \frac{|a_5|+|a_6|}{2} \cdot Ver_{Anz} + \frac{|a_7|+|a_8|}{2} \cdot Asso_{Anz}} \tag{5.7}$$

Gleichung 5.7 zeigt eine Berechnungsvorschrift für ein normiertes Abstandsmaß. Die Faktoren für die Gewichtung der einzelnen Bestandteile des Ausgangsmodells werden aus dem Durchschnitt der Beträge der Gewichtungsfaktoren für die korrespondierenden Operationen im Delta-Modell gebildet. Im Zähler der Gleichung ist zu erkennen, dass die Operationen für das Setzen von Parametern oder die Konfiguration der Komponenten in der Berechnung nicht berücksichtigt werden. Diese Änderungen sind nicht struktureller Natur und wenn sie berücksichtigt werden, muss die Anzahl aller Ports des Ausgangsmodells berücksichtigt werden, damit die Aussagekraft nicht verfälscht wird.

Alternativ können für den Vergleich von zwei oder mehr Delta-Modellen zwei Maßzahlen verwendet werden. Dafür wird S in S_{add} und S_{del} geteilt. S_{add} beschreibt die gewichtete Summe der Additionsoperationen und S_{del} die der Löschoperationen. Die beiden Zahlen sind analog zu der Darstellung der hinzugefügten und gelöschten Zeilen in einem Versionsverwaltungssystem für Quellcode. Ähnlich wie die normierte Größe können S_{add} und S_{del} ebenfalls auf die gewichtete Größe des Ausgangsmodells normiert werden.

Neben der Einschätzung, ob ein System die Variante eines anderen Systems ist, kann durch die Abstandsmaße bestimmt werden, wie strukturell ähnlich sich zwei Systeme sind. Es gilt dabei allerdings zu beachten, dass die Aussagekraft auf die Struktur der Systeme bzw. auf den Lösungsraum beschränkt ist. Es wird keinerlei Aussage über die Funktionalität getroffen. Im Extremfall kann ein Delta-Modell, das das Ausgangssystem komplett löscht und etwas Neues aufbaut, die gleiche Funktionalität realisieren. Ebenso kann eine kleine Änderung der Struktur zu einer gänzlich anderen Funktionalität führen.

5.2.3 Verketten von Delta-Modellen

Im folgenden Abschnitt wird das Delta-Metamodell um einen Mechanismus zum Verketten der Modelle erweitert. Dieser Mechanismus besteht aus einer Referenz, die von einem Delta-Modell auf ein oder mehrere Delta-Modelle verweist. Diese Referenz ist in dem erweiterten UML-Modell in Abbildung 5.8 zu erkennen. Im Folgenden wird der Mechanismus beschrieben und die Verwendung anhand eines Beispiels erläutert.

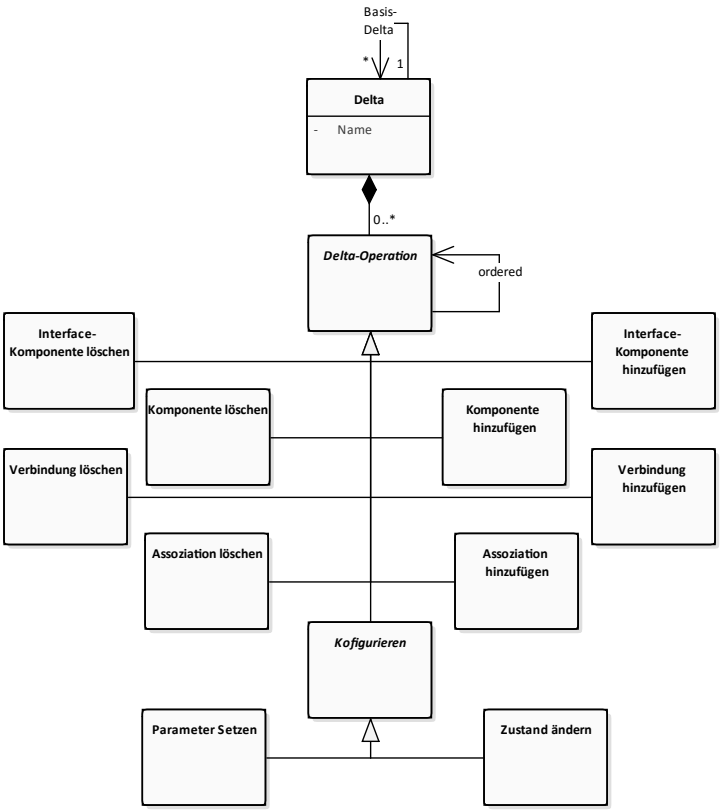


Abbildung 5.8: Delta-Metamodell mit der Erweiterung *Basis-Delta*.

Wie bereits angesprochen, beschreibt ein Delta-Modell die Transformation eines bestehenden Komponenten-Systemmodells in ein anderes. Für eine solche Transformation ist ein definierter Ausgangspunkt, auf den diese angewendet wird, erforderlich. Im vorgestellten Delta-Metamodell ist die Darstellung dieses Ausgangspunktes durch das Ergebnis einer Deltaoperation vorgesehen. Es ist zu erkennen, dass Deltas eine gerichtete Verbindung zu anderen Delta-Modellen haben können. Die Quelle dieser Verbindung ist der Ausgangspunkt für die Anwendung eines Delta-Modells an der Senke. Von einem Delta-Modell können beliebig viele Verbindungen zu anderen Delta-Modellen ausgehen. Das Ausgangsdelta wird im Weiteren als *Basisdelta* bezeichnet. Im Kontext dieser Verbindung spannen die so verbundenen Delta-Modelle einen Baum auf. Jede hinzugefügte Transformation fügt so weitere Blätter hinzu. Grundsätzlich sind zwei Situationen für ein beliebiges Delta denkbar:

1. Es existiert ein Basisdelta. In diesem Fall ist das aktuelle Delta dessen Variation.

2. Es existiert kein Basisdelta. Das jeweilige Delta-Modell repräsentiert in diesem Fall keine Variante eines Komponenten-Systemmodells. Ein solches Delta wird im Weiteren als *Root-Delta* bezeichnet.

In Abbildung 5.9 ist exemplarisch die Abhängigkeit zwischen verschiedenen Delta-Modellen dargestellt. Es sind vier Delta-Modelle (*Delta 1*, *Delta 2*, *Delta 3* und *Delta A*) sowie deren Abhängigkeiten dargestellt. Die Delta-Modelle sind jeweils mit einem Instanz-Modell assoziiert, das sich aus der Anwendung der Operationen auf die vorgesehene Startbedingung ergibt. Es ist zu erkennen, dass *Delta 1* das Root-Delta dieses Baums ist, d. h., die anderen Delta-Modelle variieren das mit *Delta 1* assoziierte Instanz-Modell. So ergänzt *Delta 2* das einfache Modul um einen Rührer nebst der für diesen notwendigen Steuerung und der Schnittstelle des erweiterten Serviceinterface des Moduls. Durch die Anwendung von *Delta A* wird die Steuerung von Hersteller X im einfachen Modul durch eine Steuerung von Hersteller Y ersetzt. Mit *Delta 3* wird das erweiterte Modul um eine zweite Pumpe ergänzt.

An diesem einfachen Beispiel wird die Bedeutung der Abhängigkeiten zwischen den Delta-Modellen deutlich. Es ist zu erkennen, dass ein Delta-Modell grundsätzlich nicht konfliktfrei auf eine beliebige Ausgangslage, die unterschiedlich zu der vorhergesehenen Ausgangslage ist, anwendbar ist. So kann *Delta 2* nicht konfliktfrei auf das mit *Delta A* assoziierte Instanz-Modell angewendet werden. Die in *Delta A* durchgeführte Änderung im Hinblick auf den Hersteller der Steuerung kann zu Konflikten bei der Übernahme führen, z. B. dann, wenn die Struktur der Steuerung gravierend verändert wird. Ein Delta-Modell ist grundsätzlich an eine gegebene Ausgangslage gebunden. Die Anwendung auf einen anderen Kontext führt möglicherweise zu einem nicht fehlerfreien Verhalten.

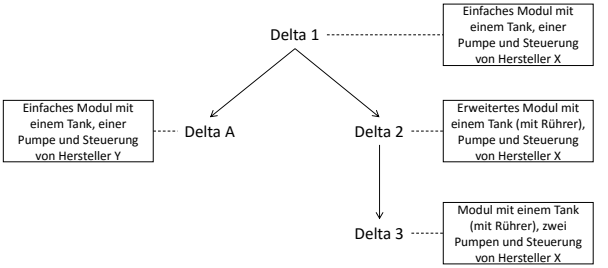


Abbildung 5.9: Beispiel für die Abhängigkeiten zwischen Delta-Modellen

Durch das Zusammenfassen von Delta-Modellen kann es zu inversen oder zu widersprüchlichen Operationen in einem Modell kommen. Durch die Beachtung einer Reihenfolge der Operationen entstehen keine inkonsistenten Modelle. Allerdings befinden sich im Modell unnötige Artefakte. Diese können jedoch leicht durch Vergleich des Gegenstands, auf den die Operation angewendet wird, identifiziert und entsprechend behoben werden. Inverse Operationen (Anlegen und Löschen eines Elements) können aus dem Modell entfernt werden. Wird eine Komponente gelöscht und neu angelegt, kann dies zu einem Ändern der Parameter reduziert werden.

5.2.4 Visualisierung

Für die Visualisierung der Delta-Modelle bietet sich eine von der Versionsverwaltung von Quellcode abgeleitete Form der Darstellung an. Dabei werden geänderte Elemente des Modells farblich im Kontext des Ergebnisses des angewandten Delta-Modells hervorgehoben. Durch die Delta-Modelle können drei Grundoperationen dargestellt werden (Hinzufügen, Löschen und Modifizieren), weswegen drei Farben für die Kodierung benötigt werden. Durch das Delta-Modell hinzugefügte Elemente sollen grün, gelöschte Elemente rot und modifizierte Ports und Zustände gelb hervorgehoben werden. So entsteht eine Darstellung des neuen Systems im Vergleich zum alten.

Die Darstellung der Komponenten des Systems kann prinzipiell auf zwei Arten erfolgen. Einerseits können die Komponenten in der Darstellungsform der jeweiligen Domäne abgebildet werden. Bei Hardware aus der Prozessindustrie ist die IEC 62424 [IEC16] ein Beispiel dafür. Nachteilig an dieser Herangehensweise ist die Vermischung von verschiedenen Darstellungsformen in einer Abbildung. Dies fordert vom Nutzer die Kenntnis einer großen Bandbreite von Darstellungsformen. Werden die Komponenten abstrakt als Bausteine dargestellt, kann der Typ über einen Namen im Kopf des Bausteins angegeben werden. Diese Darstellung als Blockschaltbild ist vielfach gebräuchlich und kann so von Anwendern mit unterschiedlichem Hintergrund verwendet werden. Externe Ports werden am Rand der Darstellung abgebildet, wie es auch in CFC's üblich ist. Die Darstellung von Dienstschnittstellen erfolgt über spezielle Blöcke, die die entsprechenden Dienstaufrufe oder die angebotenen Dienste als Block mit der dazugehörigen internen Schnittstelle darstellen.

Wenn es im Einzelfall benötigt wird, kann das jeweilige Modell durch eine domänenspezifische Darstellung wiedergegeben werden. Die vorgestellte farbliche Kodierung bleibt in diesem Fall gleich.

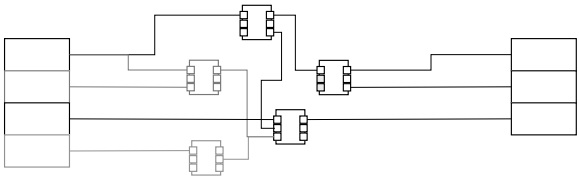


Abbildung 5.10: Beispiel für die Visualisierung von Delta-Modellen

In Abbildung 5.10 ist das Beispiel einer Visualisierung eines Delta-Modells dargestellt. Es ist die angesprochene Darstellung als Blockschaltbild zu erkennen. Sowohl das Interface als auch die einzelnen Verbindungen zwischen den Komponenten sind entsprechend ihrer Bedeutung im Delta-Modell eingefärbt. Elemente, die durch das Delta-Modell nicht verändert wurden, sind schwarz, hinzugefügte Elemente grün, gelöschte rot und Ports, deren Wert geändert sind, gelb eingefärbt.

Zur Dokumentation bzw. Veranschaulichung der Wirkung der Anwendung eines Delta-Modells können alle Komponenteninstanzen, gelöschte und hinzugefügte, nach dem Anwenden eines Delta-Modells, in einem Komponentensystem zusammengeführt werden. Zur

Interpretation ob eine Komponenteninstanz hinzugefügt oder gelöscht wurde, wird diese farblich abgebildet. Hinzugefügte Komponenten sind grün, gelöschte rot und veränderte gelb dargestellt.

5.2.5 Mapping in den Problemraum

Die reine Betrachtung von Varianten im Lösungsraum, d. h. von Delta-Modellen, ist im Kontext von Implementierungen sehr gut geeignet. Allerdings ist dieser Ansatz mit einer hervorragend geplanten und gebauten Stadt vergleichbar, für die keine Stadtpläne oder Telefonbücher verfügbar sind. Geschäfte und Häuser sind vorhanden, allerdings sind diese nicht auffindbar. Aus Sicht des Entwicklers (im Beispiel der Architekt oder Bauleiter) stellt sich der Zustand so dar, wie er sein soll: Er weiß, was implementiert ist und wie es genutzt werden kann. Für einen Nutzer, der das bestehende System verwenden möchte, gilt dies nicht. Er benötigt Pläne und Beschreibungen, um das System nutzen zu können.

Übertragen auf die Delta-Modelle betrachtet ein Nutzer (sei es Mensch oder Maschine) die Varianten unter funktionalen Gesichtspunkten. Der innere Aufbau ist dafür nicht relevant. Vielmehr muss deutlich werden, was eine Implementierung kann und wie sie zu nutzen ist.

Eine Möglichkeit der Lösung ist, ein Mapping in den Problemraum (vgl. Abschnitt 4.3.2) zu schaffen, da dort die Fähigkeiten und Merkmale eines Produkts modelliert sind. Innerhalb des Problemraums stehen die Fähigkeiten bzw. Funktionen der Produkte und nicht ihre Realisierung im Fokus. Im Folgenden wird ein Feature-Modell vorgestellt, mit dem Produkttypen auf Basis von Features beschrieben werden können. Diese modellierten Features werden mit den Delta-Modellen im Lösungsraum verbunden und ergeben so eine durchgängige Beschreibung der implementierten Funktionalität. Die durch ein Delta-Modell beschriebene Änderung am Aufbau eines Produkts (Instanz-Modell) hat im Allgemeinen Auswirkungen auf dessen Features. Welche Features davon betroffen sind, wird durch die Verbindung zwischen Delta-Modell und den zugehörigen Features abgebildet.

Das Modell ist in Abbildung 5.11 dargestellt. Kern ist der *Produkttyp*, der sich aus den *ProduktFeatures* zusammensetzt. *ProduktFeatures* sind eine lokale Abbildung von allgemein definierten *Features*. Zwischen *ProduktFeatures* können Abhängigkeiten modelliert werden. Etwa, dass ein anderes *ProduktFeature* für die Nutzung eines bestimmten *ProduktFeatures* erforderlich ist oder dieses ausschließt. Von den *Delta-Modellen* existiert jeweils ein Verweis auf die realisierten *ProduktFeatures*. Durch die aufgespannten Assoziationen sind Delta-Modelle aus einer funktionalen Perspektive auffindbar.

In der praktischen Anwendung erhält der Nutzer eine Menge von Produkttypen, die jeweils von Features beschrieben werden. Über die Produkttypen und Features kann ein Produkt mit den benötigten Features gesucht werden. Das Ergebnis der Suche beinhaltet die Produkttypen, auf die die geforderten Kriterien zutreffen. So können geeignete Delta-Modelle identifiziert werden, um das gewünschte Produkt zu erzeugen.

Die genaue Definition der Features ist spezifisch für den jeweiligen Anwendungsfall. Im Rahmen des Projekts BaSys 4.0 wurde mit dem Aufbau eines Feature-Modells begonnen.

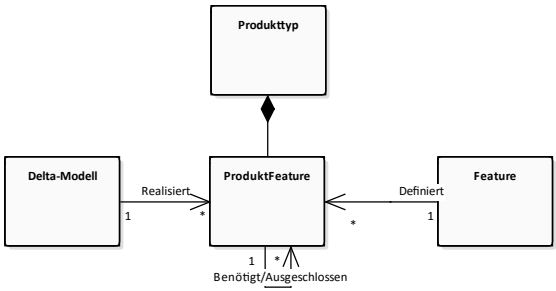


Abbildung 5.11: Feature-Modell für die Darstellung im Problemraum

5.3 Gesamtkonzept für die variantenbasierte Wiederverwendung

Im folgenden Abschnitt wird das Gesamtkonzept für die variantenbasierte Wiederverwendung vorgestellt. Grundlage dafür sind das Komponenten- und das Delta-Modell aus den vorangegangenen Abschnitten. Als erstes wird ein Überblick über das Konzept gegeben und die darin enthaltenen Elemente erläutert. Anschließend wird die durch das Konzept vorgenommene Unterscheidung zwischen Versionen und Varianten vorgestellt. Nach der Erläuterung der im Modell enthaltenen Transformationen wird erörtert, was durch das Konzept wiederverwendet wird. Für den praktischen Einsatz wird anschließend eine verteilte Architektur vorgestellt. Abschließend werden Prozesse für die Nutzung in der Praxis vorgeschlagen.

5.3.1 Überblick über das Konzept

Die vorgestellten Modelle können rein deskriptiv verwendet werden. Um diese jedoch in der Praxis möglichst nutzbringend anzuwenden, ist die Einbindung in bestehende Engineering-Prozesse erforderlich. Dafür ist die Verknüpfung der Modelle durch Transformationen und deren geeignete Nutzung in Prozessen notwendig. In diesem Abschnitt wird die Verwendung der unterschiedlichen Modelle, deren Verknüpfung zur physischen Welt und die Transformationen erläutert.

Ein Überblick über das Konzept ist in Abbildung 5.12 dargestellt. Diese Darstellung ist eine Sicht auf die Modelle und physische Welt sowie die Verbindungen und Transformationen zwischen den Modellelementen. Es sind das Delta-Modell, das Instanz-Modell und das Typ-Modell zu erkennen. Auf der rechten Seite ist die physische Welt mit den Komponententypen und den aus Komponenteninstanzen zusammengesetzten Systemen dargestellt. Die Darstellung unterteilt sich so in die Modellwelt und die reale Welt. Ein Überblick über die in Abbildung 5.12 verwendeten Elemente ist in Tabelle 5.1 enthalten.

Die Pfeile (rot gestrichelt) zwischen den Modellen und den Komponenten stellen die Transformationen dar, die zur Erzeugung der Modelle bzw. zu deren Umwandlung erforderlich sind. Die schwarzen durchgezogenen Pfeile stellen Verweise bzw. Abhängigkeiten zwischen den Elementen des Bilds dar. So ist der Verweis von einem Delta-Modell auf sein Basis-Delta-Modell dargestellt. Dies führt zu der diskutierten Baumstruktur innerhalb der Delta-Modelle. Analog zu den Komponentensystemen, die Instanziierungen der Komponententypen sind, besteht eine Instanziierungsbeziehung zwischen dem Instanz-Modell und dem Typ-Modell.

Name des Elements	Beschreibung
Komponententypen	Typen von Komponenten, die zur Verwendung in Komponentensystemen vorgesehen sind. Dies können Klassen, Prototypen oder Beschreibungen von Komponenten sein.
Typ-Modelle	Modelle der Komponententypen (vgl. Kapitel 5.1).
Komponentensysteme	Aus Instanzen der Komponententypen zusammengesetzte Systeme. Diese stellen Implementierungen der Instanz-Modelle dar. Abhängig vom betrachteten Kontext kann es sich hierbei um Hardware, Software oder hybride Systeme handeln.
Instanz-Modelle	Modelle von Komponentensystemen auf Basis der vorhandenen Typ-Modelle (vgl. Kapitel 5.1).
Delta-Modelle	Modelle, die die Transformation von einem Instanz-Modell in ein anderes beschreiben. Diese liegen als Objektstrukturen vor (vgl. Kapitel 5.2).

Tabelle 5.1: Übersicht über die Elemente des Konzepts

Eine sehr nützliche Verbindung ist die Referenz zwischen den Komponententypen im Typ-Modell und den konkreten Komponententypen. Diese kann über einen Verweis oder einen Link realisiert werden. Sie ist eine 1:n Beziehung, mit der beschrieben wird, welche Typen von Komponenten den Komponententyp im Modell realisieren. Über diese Verbindung kann auch die Versionierung von Komponenten abgebildet werden. Solange das Interface der Komponente kompatibel zum modellierten Interface ist und die realisierte Funktionalität gleich ist, können andere Versionen der Komponente ebenfalls den Komponententyp im Modell realisieren. Aus dem Modell heraus ist der Verweis auf alle kompatiblen Realisierungen möglich. Die *verwendet* Referenz zwischen den Delta-Modellen und den Typ-Modellen bedeutet, dass die Delta-Modelle bei Hinzufügen-Operationen auf die durch die Typ-Modelle beschriebenen Komponenten verweisen. Diese Komponenten stellen die „Toolbox“ dar, die durch die Delta-Modelle verwendet werden kann.

Für die Verfolgung der eingesetzten Instanzen ist die Realisierung der Verbindung zwischen den Komponententypen und -instanzen erforderlich. Diese Verbindung muss explizit sein. Eine implizite Verbindung, wie sie beispielsweise beim Kopieren und Einfügen entsteht, ist für die Nachverfolgbarkeit nicht sinnvoll. Je nach vorhandenen Randbedingungen kann sie als bi- oder unidirektionale Verbindung umgesetzt werden. Die Verfolgung der Instanzen ermöglicht im weiteren Verlauf deren Aktualisierung auf eine neue Version. Ebenso ist es möglich, neue Varianten von Komponentensystemen anstelle von älteren zum Einsatz zu bringen.

In Tabelle 5.2 ist eine Übersicht über die in Abbildung 5.12 dargestellten Transformationen und deren Ergebnisse aufgeführt. In der Spalte *Vorbedingung* sind die jeweils für die Transformation benötigten Elemente festgehalten. Wenn die dargestellte Ausgangssituation in einem konkreten Anwendungsfall nicht eingehalten wird, kann die Transformation nicht durchführbar sein oder zu einem nicht definierten Ergebnis führen. Eine Implementierung des Konzepts muss dies entsprechend berücksichtigen und das Vorhandensein einer gültigen Vorbedingung prüfen. Die Transformationen werden im Einzelnen in Kapitel 5.3.2 beschrieben.

Name der Transformation	Vorbedingung	Ergebnis
erzKomponentenTypModell	Komponententypen oder Beschreibung liegen vor	Typ-Modell
erzKomponentenInstModell	Komponentensystem und Typ-Modelle der verwendeten Komponententypen liegen vor	Instanz-Modell
erzImpl	Instanz-Modell, Typ-Modelle und die zugehörigen Implementierungen liegen vor	Komponentensystem
erzDeltaModell	Zwei Instanz-Modelle liegen vor	Delta-Modell
erzInstanzModell	Delta-Modell und Instanz-Modell als Ausgangspunkt für dessen Anwendung liegen vor	Instanz-Modell

Tabelle 5.2: Übersicht über die Transformationen des Konzepts

Versionen und Varianten

Im Rahmen des Konzepts wird zwischen Versionen und Varianten unterschieden. Varianten sind unterschiedliche Komponentensysteme, die sich entsprechend der vorgestellten Definition ähnlich sind (vgl. Abschnitt 4.3.1). Diese Varianten bzw. die Unterschiede zwischen ihnen werden mittels der Delta-Modelle beschrieben.

Zusätzlich dazu können – orthogonal zu den Varianten – von den verwendeten Komponententypen unterschiedliche Versionen existieren. Versionen müssen dafür die Funktionalität und die Schnittstelle des Typ-Modells einer Komponente realisieren. In beiden Fällen (Funktionalität und Schnittstelle) müssen die Versionen die Anforderungen des Modells mindestens erfüllen. Eine Übererfüllung, z. B. durch ein längeres Interface oder zusätzliche Funktionalität, ist kompatibel und wird ebenso abgebildet.

Diese Interpretation von Versionen im Konzept ist anders als die zeitliche Fortschreibung von Implementierungen einer Anwendung, die in der Literatur verwendet wird. In dieser klassischen Sicht entsteht eine neue Version einer Anwendung durch eine Änderung ihrer Implementierung. Je nach Größe der Änderung wird die neue Versionsbezeichnung gewählt. Bei großen Änderungen ist die neue Version die nächste ganze Zahl. Ansonsten wird die Nachkommastelle inkrementiert. Eine Betrachtung der Funktionalität der Anwendung und ihrer Kompatibilität gegenüber der Umgebung findet nicht statt. Diese funktionale

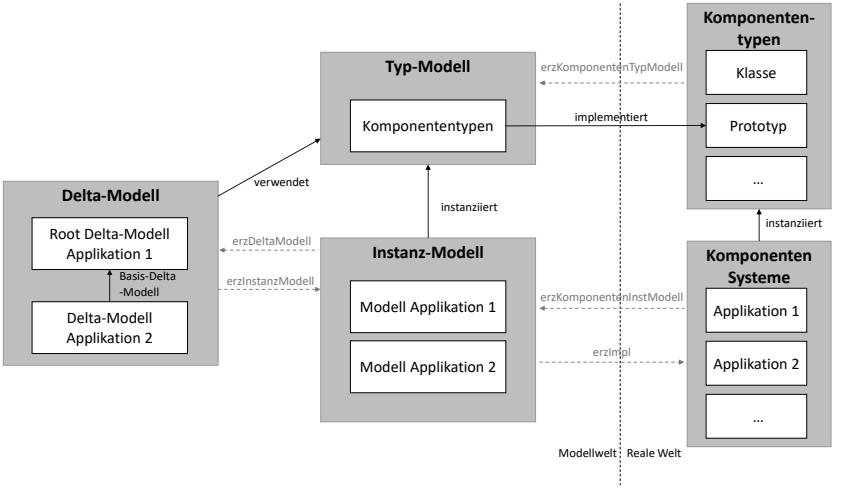


Abbildung 5.12: Das Konzept in der Gesamtsicht

Betrachtung steht für das Konzept im Fokus. Daher wird der Versionsbegriff auf das Typ-Modell der Komponenten angewendet. So wird von der Ähnlichkeit der Implementierung abstrahiert und der Schwerpunkt auf die Funktionen und Schnittstellen gelegt.

So können auch unterschiedliche Implementierungen, die die im Modell beschriebenen Voraussetzungen (Schnittstellen und Funktionalität) erfüllen, als Versionen eines Typ-Modells verwendet werden. Dies ist vorteilhaft, da die Funktionalität und die Kompatibilität für den Anwender wichtiger ist als die Einordnung in eine Implementierungshistorie. Diese bleibt für den Entwickler weiter bestehen. In der Entwicklung der Komponententypen ist die Nutzung dieser Verweise sinnvoll und richtig. In der Weiterentwicklung von Implementierungen ist der Zusammenhang zwischen unterschiedlichen Versionen einer Implementierung wichtig, um an geeigneten Stellen Änderungen durchführen zu können. Für den Anwender ist dagegen der vorgestellte erweiterte Versionsbegriff zielführender.

Allein die Verwendung von unterschiedlichen Versionen eines Komponententyps führt nicht zur Entstehung einer neuen Variante. So kann ein Komponententyp auf unterschiedlichen Systemen durch unterschiedliche Komponenten realisiert werden. Trotzdem ändert sich hierdurch die Variante nicht. Erst die Nutzung eines anderen Komponententyps oder eine veränderte Anbindung an die Umgebung führt zu einer neuen Variante und damit zu einem Delta-Modell.

5.3.2 Modelltransformationen

Im folgenden Abschnitt werden die in Abbildung 5.12 und Tabelle 5.2 dargestellten Transformationen näher beschrieben. Die Vorbedingungen und das Ergebnis der jeweiligen

Transformation ist in Tabelle 5.2 zu finden.

erzKomponentenTypModell

Ziel dieser Transformation ist die Erzeugung des Modells eines Komponententyps. Die genaue Umsetzung der Transformation ist abhängig von der Art der vorliegenden Beschreibung des Komponententyps. Liegt diese in einer Form vor, die durch einen Computer auswertbar ist, so kann eine Applikation entwickelt werden, die diesen Prozess automatisiert. Ist dies nicht der Fall, da beispielsweise nur der Prototyp einer Hardwarekomponente vorliegt, so muss das Modell manuell erzeugt werden (vgl. Kapitel 5.1). Das Vorgehen ist unabhängig von der konkreten Umsetzung gleichartig.

Im ersten Schritt muss der Typ der Komponenten identifiziert werden. Dieser kann beim Anlegen des Objekts, das den Komponententyp repräsentiert, in den Objektnamen integriert werden. Mindestens wird der Verweis auf den Komponententyp im Objekt erzeugt. Dies kann entweder als unidirektionaler Verweis, beispielsweise ein Bezeichner, erfolgen oder, wenn es die verwendete Implementierung und Komponente zulassen, eine bidirektionale Referenz sein. In diesem Schritt muss auch die Funktionalität des abgebildeten Komponententyps im Modell abgelegt werden. Dies kann textuell erfolgen. Allerdings bietet ein einheitliches Verzeichnis, das analog zu Merkmalen die Funktionalitäten oder Fähigkeiten beschreibt, einige Vorteile bezüglich der Eindeutigkeit und Auffindbarkeit dieser Zuordnungen (vgl. Kapitel 5.1).

Als letztes wird das Interface der Komponente nachgebildet. Dafür werden die Ein- und Ausgangsports der Komponente mit deren jeweiligen Typen angelegt. Die Ports werden über die Zuordnung im Namensraum, d. h. über Namensgleichheit, eindeutig identifiziert. Die Namen können auch über mehrere Versionen einer Komponente als konstant betrachtet werden, da durch vorherrschende Namenskonventionen eine Beschreibung des Ports über den Namen vorgenommen wird. Sollte sich der Name von Ports ändern, so handelt es sich vom Standpunkt des Komponenten-Modells um eine Komponente mit der gleichen Funktionalität aber einem nicht kompatiblen Interface. Dies führt zur Ausprägung eines neuen Komponententyps bzw. bei Verwendung dieses Komponentyps zu einer neuen Variante.

erzKomponentenInstModell

Das Ziel dieser Transformation ist die Erzeugung eines Instanz-Modells aus einem vorliegenden Komponentensystem unter Verwendung von Komponententypmodellen. Ausgangspunkt ist ein bestehendes Komponentensystem. Ebenso wie bei der Erzeugung der Typ-Modelle ist der eigentliche Prozess unabhängig davon, ob er manuell oder (teil-)automatisiert durchgeführt wird. Der verwendete Automatisierungsgrad hängt im Wesentlichen davon ab, inwieweit das vorliegende System automatisiert erkundbar ist. Dass das System erkundbar ist, ist insbesondere bei Systemen aus Softwarekomponenten wie Funktionsbausteinnetzwerken zu erwarten.

Im ersten Schritt wird der Name des Komponentensystems ermittelt und ein Objekt für seine Repräsentation entsprechend dem Komponenten-Metamodell angelegt. Anschließend

wird über die Komponenten des Systems (interne Komponenten und Interfacekomponenten) iteriert. Wenn eine aufgefundene Komponente im Komponententypmodell enthalten ist, wird ein Modell im Instanz-Modell instanziiert. Die Identifizierbarkeit wird über die Namensgleichheit im Namensraum sichergestellt. Liegt zu einer Komponente kein passendes Typ-Modell vor, muss dies protokolliert werden. Wenn möglich, kann der Abgleich mit anderen Typ-Modellen durchgeführt und so ein passendes Modell gefunden werden. Ist das nicht möglich, muss der Nutzer manuell eingreifen. Es ist zusätzlich zu erörtern, ob und wenn ja, wie viele Hierarchiestufen des Komponentensystems im Instanz-Modell nachgebildet werden sollen. Insbesondere die Identifikation von eingebetteten Strukturen ist dabei interessant, um diese für die Wiederverwendung zu nutzen. Damit sind Strukturen von Komponenten gemeint, die in verschiedenen Systemen wiederverwendet werden. Diese liegen als Prototyp vor und sind aus Basiskomponenten zusammengesetzt. Für die Nachbildung der unterschiedlichen Hierarchiestufen sind drei Möglichkeiten denkbar:

1. Nachbildung des Instanz-Modells mit der Erkundungstiefe eins
   Bei dieser Möglichkeit wird das gegebene System nachgebildet, ohne dabei geschachtelte Komponenten zu erfassen. Aggregierte Komponenten werden als Blackboxen betrachtet, d. h., wenn sie unbekannt sind, findet keine Identifikation statt.

2. Beliebige Erkundungstiefe ohne Identifikation von Strukturen
   Das vorliegende Komponentensystem wird in einer beliebigen Erkundungstiefe nachgebildet, ohne dabei jedoch eingebettete Strukturen aus dem Typenmodell zu suchen und zu erkennen. Die verschachtelten Strukturen werden aus den vorhandenen Basiskomponenten modelliert.

3. Rekursive Identifikation von eingebetteten Strukturen
   Dieser Ansatz erfasst auch alle Komponenten des Systems. Während der Erfassung wird allerdings versucht, aggregierte Komponenten zu erkennen und das Instanz-Modell unter Verweis auf den jeweiligen Prototyp aufzubauen.

Die drei vorgestellten Möglichkeiten unterscheiden sich hinsichtlich der Komplexität ihrer Umsetzung. Sie sind in aufsteigender Komplexität sortiert. Zunächst wird die iterative Erkundung und die Identifikation von eingebetteten Strukturen hinzugefügt. Gleichzeitig steigt der Nutzen der Wiederverwendung an, wenn Strukturen identifiziert und auf einen gemeinsamen Prototyp zurückgeführt werden können. So wird die doppelte Wartung von Implementierungen vermieden und Aktualisierungen von Implementierungen können an alle Instanzen propagiert werden.

Wenn eine aggregierte, vorher noch nicht verwendete, Komponente gefunden wird, kann diese direkt über einen Prototypen der Liste der Komponententypen hinzugefügt und anschließend verwendet werden. Handelt es sich um eine atomare Komponente, die nicht Bestandteil des Typ-Modells ist, muss das Fehlen des benötigten Komponententyps protokolliert werden.

erzImpl

Diese Transformation kann als eine Invertierung von *erzKomponentenInstModell* betrachtet werden. Sie wandelt ein Instanz-Modell in eine konkretes Komponentensystem um.

Durch die Existenz von mehreren Implementierungen eines Komponententyps (z. B. durch mehrere Versionen) ist diese Transformation nicht zwingend eindeutig.

Im Rahmen der Transformation wird durch das vorliegende Instanz-Modell iteriert und die Realisierungen der verwendeten Komponenten ermittelt. Um die Realisierungen zu finden, wird der Typ der Komponente im Typ-Modell gesucht. In diesem Modell sind Verweise auf kompatible Implementierungen enthalten. Nach deren Auffinden werden diese im Komponentensystem instanziiert. Je nach Art des Komponentensystems kann es sich um die Instanziierung einer Softwarekomponente handeln oder um das Hinzufügen einer Hardwarekomponente.

erzDeltaModell

Ziel dieser Transformation ist es, durch den Vergleich zweier Instanz-Modelle ein Delta-Modell zu erzeugen. Voraussetzung für die Transformation ist das Vorliegen zweier Instanz-Modelle. Der Grad ihrer Überschneidung ist für die Durchführung der Transformation unerheblich, da es sich im Extremfall um einen vollständigen Austausch der Modelle handelt.

Der Vergleich der Modelle kann unter Berücksichtigung von unterschiedlichen Randbedingungen durchgeführt werden. Eine Auffindung von gleichen Komponenten kann über eine Gleichheitsprüfung hinsichtlich des Komponententyps und/oder des Namens der Komponente erfolgen. Darüber hinaus kann die Position der Komponente innerhalb des modellierten Komponentensystems als Vergleichsgröße herangezogen werden. Die Position besteht dabei aus Verbindungen und verbundenen Komponenten. Die Namen von Komponenten folgen für eine bessere Verständlichkeit von Komponentensystemen einer gewissen Systematik (vgl. PLT-Stellenbezeichnungen [IEC16]). Daher reicht es für die Mehrzahl der Anwendungsfälle aus, die Namen zweier Komponenten als Vergleichsgröße heranzuziehen. Die verwendete Vergleichssystematik kann ohne Beeinträchtigung des restlichen Konzeptes angepasst und verfeinert werden.

Nach der erfolgten Detektion eines Unterschiedes zwischen den Instanz-Modellen wird ein entsprechendes Objekt im Delta-Modell erzeugt. In dem Objekt werden die entsprechenden Parameter gespeichert. Dies können z. B. die manipulierte Komponente, der zu setzende Parameter, dessen neuer Wert oder der Name und der Typ der zu erzeugenden Komponente sein.

erzInstanzModell

Ausgehend von einem Delta-Modell wird durch diese Transformation das entsprechende Instanz-Modell erzeugt. Entweder handelt es sich bei dem Delta-Modell um ein Root-Delta oder um ein Delta-Modell mit beliebig vielen Basisdeltas. Liegt ein Root-Delta vor, können die enthaltenen Operationen direkt ausgeführt werden. Ist das vorliegende Delta-Modell kein Root-Delta, muss zunächst das Basisdelta rekursiv vom zugehörigen Root-Delta ausgehend erzeugt werden. Für diese Erzeugung gibt es zwei Optionen: Zum einen die echte Rekursion, wobei jedes Delta-Modell auf dem Pfad vom Root-Delta bis Basisdelta nacheinander angewendet wird, und zum anderen das Zusammenfassen aller Operationen zu einem Delta-Modell. Die Anwendung des Gesamt-Delta-Modells kann optimiert werden, indem

es auf Überschreibungen des gleichen Parameters von unterschiedlichen Operationen oder auf das Vorliegen von Hinzufüge- und Löschen-Operationen mit der gleichen Komponente als Ziel überprüft wird.

5.3.3 Gegenstand der Wiederverwendung

Ziel der Arbeit ist ein Konzept für die Wiederverwendung von Teillösungen in komponentenbasierten Architekturen. Dieses Konzept soll möglichst unabhängig von der konkreten Implementierung der Komponenten sein. Nachdem die Modelle für das Konzept und die Transformationen zwischen den Modellelementen diskutiert wurden, wird im Folgenden die Fragestellung behandelt, was durch das Konzept wiederverwendet wird.

Für das Austauschen von Komponentensystemen kann das zugrundeliegende Meta-Metamodell vereinheitlicht werden. Dies steht jedoch im Gegensatz zur Forderung, dass Bestandssysteme durch das Konzept berücksichtigt werden müssen. Anstatt das Meta-Metamodell zu verändern, wird das dieser Arbeit zugrunde liegende Komponenten-Modell für die Abstraktion von dem konkreten Komponentensystem verwendet. Die Komponentenstrukturen werden von ihrer Realisierung entkoppelt und die in den Strukturen enthaltene Information kann separat genutzt werden. So wird ein Austausch der implementierungsunabhängigen Systemstrukturen ermöglicht, ohne die Bestandssysteme verändern zu müssen.

$$InstanzModell_1 + \Delta_A = InstanzModell_2 \tag{5.8}$$

Die Möglichkeiten der Deltamodellierung erlauben es, Wiederverwendung auf zwei Arten zu betreiben. In Gleichung 5.8 ist der Zusammenhang zwischen zwei Instanz-Modellen und einem Delta-Modell Δ_A dargestellt. Es ist zu erkennen, dass das $InstanzModell_1$ ebenso mit einem anderen Delta-Modell verwendet werden kann. In diesem Fall ist das Instanz-Modell Gegenstand der Wiederverwendung. Alternativ kann Δ_A auf ein anderes Instanz-Modell angewendet werden und stellt in diesem Fall den Gegenstand der Wiederverwendung dar.

In Abbildung 5.13 sind die beiden Arten der Wiederverwendung im Überblick dargestellt. Auf der linken Seite ist zu erkennen, wie das Delta-Modell als Gegenstand der Wiederverwendung benutzt wird. Die durch das Delta-Modell beschriebene Struktur kann in unterschiedlichen Kontexten, d. h. Anwendungsumgebungen, angewendet werden. Die zweite Art ist auf der rechten Seite zu erkennen. Der Kontext, d. h. die ursprüngliche (Teil-)Lösung, wird für unterschiedliche Delta-Modelle verwendet. Im einem Fall ist das Delta-Modell Gegenstand der Wiederverwendung und im anderen Fall die Umgebung, in die die Delta-Modelle eingefügt werden.

Die Anwendung von Delta-Modellen in einem anderen als dem ursprünglich vorgesehenen Kontext ist nicht zwingend konfliktfrei möglich. Insbesondere nicht vorhandene Randbedingungen, wie nicht existierende Komponenten oder Interfacekomponenten des Systems, zu denen eine Verbindung aufgebaut werden soll, können zu Konflikten führen. Die Nutzung der Komponenten-Modelle als Abstraktion der konkreten Implementierung erlaubt die Anwendung von Delta-Modellen auf einen anderen Kontext ohne die Erzeugung einer inkonsistenten Implementierung. So kann ein Delta-Modell bewusst auf ein anderes Basisdelta angewendet werden. Anhand der Ergebnisse der Transformation, in diesem Fall

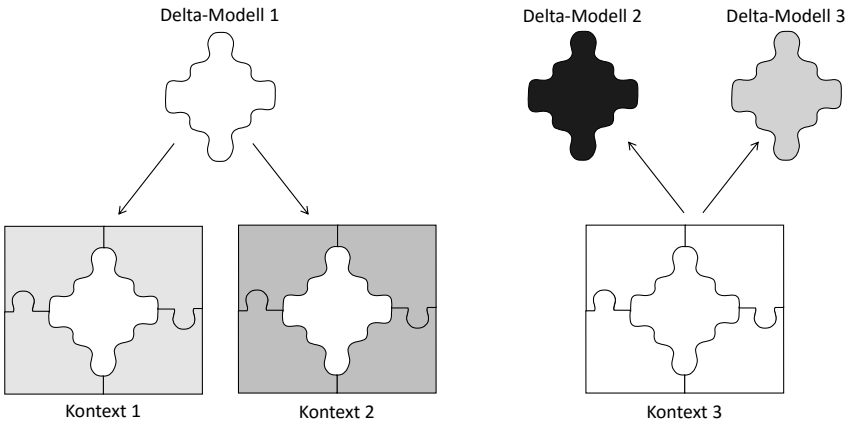


Abbildung 5.13: Arten der Wiederverwendung

insbesondere anhand der Fehlermeldungen, kann deren Qualität ermittelt werden. Nach der Ausführung der Transformation muss das Instanz-Modell auf Konsistenz geprüft werden. Dabei muss sichergestellt werden, dass die Typen von verbundenen Ports gleich sind. Die Überprüfung der Typengleichheit muss anhand der spezifischen Randbedingungen der Ports erfolgen und neben der Gleichheit mögliche Konvertierungen berücksichtigen (vgl. Tabelle für die Konvertierung von Datentypen in [IEC14b]). Zusätzlich kann eine Überprüfung, ob alle Komponenten mit Eingängen mindestens einen verbunden Eingang haben und ob bei Komponenten mit Ausgängen mindestens einer von ihnen verbunden ist, Aufschluss über mögliche Fehler der Transformation geben. Die fehlende Anbindung von Komponenten kann ein Indiz dafür sein, dass die Komponente nicht optimal in das umgebende Komponentensystem eingebunden ist und daher ein Fehler durch die Anwendung des Delta-Modells vorliegt.

5.3.4 Die verteilte Nutzung der Modelle

In den vorangegangenen Ausführungen zu den Modellen und dem Verwendungskonzept wurde der Fokus auf die Modellierung und die Funktionalität gelegt. Im folgenden Abschnitt rückt der physische Aufbau und die Verteilung der Modelle in einem System für den praktischen Einsatz in den Blickpunkt (vgl. Abschnitt 4.4.2).

In Abbildung 5.14 ist der schematische Aufbau einer dezentralen Verwendung der Modelle dargestellt. Es ist zu erkennen, dass es sich dabei um eine Server-Client-Architektur handelt. Der Server stellt die zentrale Sammelstelle für die Delta-Modelle und damit für bestehende Implementierungen dar. Die Clients sind in die jeweiligen Engineering-Umgebungen eingebettet, von denen aus die im Server gesammelten Implementierungen abgerufen und neue hinzugefügt werden. Durch die Clients wird so die dezentrale Wiederverwendung und Veränderung der Modelle ermöglicht.

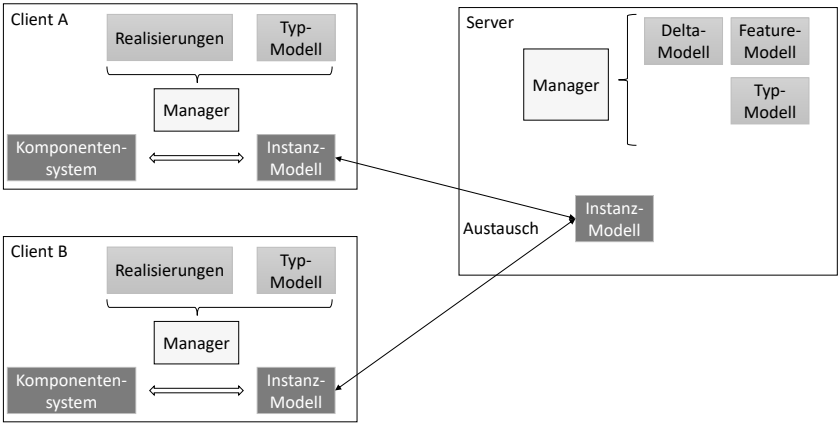


Abbildung 5.14: Verteilte Architektur für die Nutzung der Modelle

Auf dem zentralen Server werden die Delta-Modelle und die Abhängigkeiten zwischen ihnen abgelegt. Neben den Delta-Modellen werden die Modelle der Komponenten auf dem Server gespeichert. Dauerhaft werden dort jedoch ausschließlich die Typ-Modelle der Komponenten abgelegt. Die Instanz-Modelle werden entweder von einem Client auf den Server transferiert oder auf dem Server aus Delta-Modellen erzeugt. Aus der Differenz zwischen zwei Instanz-Modellen wird mittels der vorgestellten Transformationen ein Delta-Modell erzeugt. Dieses wird an der richtigen Stelle in den Delta-Baum eingefügt und mit dem Feature-Modell auf dem Server verbunden.

Die Realisierung der benötigten Dienste und die Strukturierung der Objekte und Modelle werden von einem Delta-Manager und einem Typ-Modell-Manager übernommen. Diese Manager bieten die vorgestellten Transformationen als Dienste an und bilden die Zugangspunkte für Dienstnutzer. Je nach Anwendungsfall kann es zur Verbesserung der Nutzbarkeit sinnvoll sein, die atomaren Dienste direkt in den Managern zu höherwertigen Diensten zu aggregieren und diese nach außen zur Verfügung zu stellen.

Auf dem Client befindet sich ein entsprechender Manager, der für die Umgebung bzw. für den Nutzer den Zugangspunkt für die Interaktionen bildet. Er bietet die Möglichkeit, die auf dem Server abgelegten Delta- und Feature-Modelle zu erkunden und nach vorgegebenen Randbedingungen zu durchsuchen. Wenn ein geeignetes Delta-Modell bzw. das dazugehörige Komponentensystem gefunden ist, kann es vom Server heruntergeladen werden. Vom Server wird dafür das Instanz-Modell des Komponentensystems erzeugt und auf den Client übertragen. Dort werden die Elemente des Instanz-Modells mit den lokal gültigen Implementierungen zusammengeführt und das konkrete Komponentensystem aufgebaut. Nach dem Ändern des Komponentensystems kann die neue Variante auf den Server hochgeladen und in den Delta-Baum eingefügt werden. Die Implementierungen der Komponenten bleibt auf dem jeweiligen Client. Aus dem Komponentensystem wird das dazugehörige Instanz-Modell erzeugt. Dieses dient als generisches Austauschformat für Komponentensysteme

und wird auf den Clients in die jeweilige Implementierung überführt. Dies ermöglicht, die Strukturen der Lösungen in unterschiedliche Umgebungen und Implementierungen zu überführen.

Instanz-Modelle können auf unterschiedliche Arten zwischen Server und Client ausgetauscht werden. Eine Möglichkeit zur Realisierung des Austauschs ist es, das Modell in ein gängiges Dateiformat, z. B. XML, abzubilden. Diese Datei, in der die jeweiligen Modelle enthalten sind, kann anschließend übertragen werden. Diese dateibasierte Variante bringt jedoch einen großen Mehraufwand mit und ist somit nicht sehr effizient. Alternativ kann das Modell in der jeweils anderen Laufzeitumgebung erzeugt werden, indem Dienste zum Anlegen und Manipulieren von Objekten der Laufzeitumgebungen verwendet werden. Bei dieser Übertragungsart müssen sehr viele Dienste aufgerufen werden, was einen hohen Kommunikationsaufwand bedeutet. Wegen der angesprochenen Probleme wird in der vorliegenden Arbeit die (De-)Serialisierung des Instanz-Modells durch ein gängiges Format, nämlich der JavaScript Object Notation (JSON), zur Übertragung von Objekten verwendet. Dies ermöglicht die Übertragung in einer einfachen und effizienten Weise.

Der Komponententyp im Typ-Modell kann in der physischen Welt nicht nur durch Klassen sondern auch durch Prototypen realisiert werden. Bei einem direkten Transfer einer Applikation von einer Herkunfts-Laufzeitumgebung in eine Ziel-Laufzeitumgebung müssen die Schnittstellen der Komponenten des Systems kompatibel sein. Bei der Realisierung durch eine Klasse ist eine unterschiedliche Anzahl von Eingängen in den Funktionsbaustein denkbar. Es können drei Fälle auftreten: Der implementierte Funktionsbaustein in der Ziel-Laufzeitumgebung besitzt eine kleinere, eine größere oder eine gleich große Schnittstelle als der in der Herkunfts-Laufzeitumgebung. Ist die Schnittstelle größer oder gleich groß, so ist die Implementierung des Funktionsbausteins ohne Änderung kompatibel zum Typ-Modell. Bei einer kleineren Schnittstelle ist die Implementierung nicht mehr kompatibel zum Typ-Modell. Dadurch ist eine Übertragung der Komponentensysteme, die diese inkompatiblen Bausteine verwenden, nicht möglich. Durch die Abstraktion mit dem in dieser Arbeit vorgestellten Komponenten-Modell besteht die Möglichkeit, den Komponententyp durch einen Prototypen zu realisieren. Somit kann die geforderte Funktionalität durch ein Netzwerk von Funktionsbausteinen in der Ziel-Laufzeitumgebung realisiert werden. Der Unterschied einer Realisierung durch Klassen zu der durch einen Prototypen wird im folgenden Beispiel deutlich: Das Addieren von Werten wird typischerweise durch eine Funktionsbaustein-Klasse realisiert. Ist in der Ziel-Laufzeitumgebung lediglich ein Funktionsbaustein mit zwei Eingängen realisiert, in der Herkunfts-Laufzeitumgebung jedoch einer mit drei Eingängen, so kann der Funktionsbaustein und somit das Komponentensystem nicht mehr übertragen werden. Wird das Addieren aber in der Ziel-Laufzeitumgebung als Protoyp realisiert, so kann durch eine geschickte Verschaltung von zwei Addierer-Bausteinen die zu kleine Schnittstelle kompensiert werden. Das Beispiel ist in Abbildung 5.15 exemplarisch dargestellt.

Dieses einfache Beispiel zeigt, wie durch die Verwendung einer zusätzlichen Abstraktionsschicht die Kompatibilität und Wiederverwendbarkeit von komponentenbasierten Systemen auch bei zunächst inkompatiblen Komponenten erreicht werden kann. Dies kann analog ebenso auf Hardwarekomponenten angewendet werden. Entspricht beispielsweise der Durchsatz einer Pumpe nicht den geforderten Spezifikationen, so kann dies durch die parallele Verwendung von zwei oder mehr Pumpen kompensiert werden.

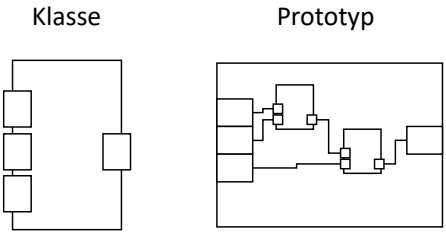


Abbildung 5.15: Realisierung einer Komponente zur Addition von drei Werten durch eine Klasse und durch eine Verschaltung von zwei Komponenten, die lediglich zwei Werte addieren können.

5.3.5 Verwendung in der Praxis

Im folgenden Abschnitt werden exemplarisch Abläufe für die Nutzung des vorgestellten Konzeptes in der Praxis vorgestellt. Für diese Abläufe werden die in Abschnitt 5.3.2 beschriebenen Transformationen genutzt. Zu jedem Ablauf wird die Ausgangssituation, der eigentliche Prozess und das Ergebnis beschrieben.

Es wird in den folgenden Ausführungen davon ausgegangen, dass ein verteiltes System mit Clients und ein zentraler Server für die Verwaltung der Delta-Modelle vorliegt. Sowohl die Clients als auch der Server können durch Laufzeitsysteme realisiert werden. Die Komponentensysteme sind hybride Systeme, d. h. Hard- und Softwaresysteme.

Von der Implementierung zum Delta-Modell

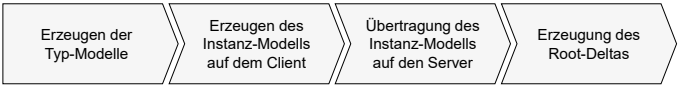


Abbildung 5.16: Ablauf zur Erzeugung eines Delta-Modells

Ausgangssituation: Es existieren mindestens ein Komponentensystem, ein Client für den Aufbau des Instanz-Modells und ein zentraler Server, auf dem die Delta-Modelle der Implementierungen abgelegt werden. Auf dem Server und dem Client sind die jeweiligen Manager vorhanden und betriebsbereit.

Ablauf: Eine Übersicht des Ablaufs ist in Abbildung 5.16 dargestellt. Der erste Schritt ist die Erzeugung der Typ-Modelle aus den Implementierungen oder einer Typbeschreibung. Dafür wird die Transformation *erzKomponentenTypModell* verwendet. Die Typ-Modelle werden sowohl auf dem Server als auch auf allen Clients erzeugt, sodass eine einheitliche Basis von Typ-Modellen vorliegt. Im zweiten Schritt wird das Instanz-Modell durch die Transformation *erzKomponentenInstModell* auf einem Client erzeugt. Das erzeugte

Instanz-Modell wird anschließend auf den Server übertragen. Danach kann das Modell auf dem Client gelöscht werden. Falls das Instanz-Modell Variante eines existierenden ist, wird das Delta-Modell unter Verweis auf ein Basis-Delta erzeugt. Ist dies nicht der Fall, wird das Delta-Modell und ebenso die Verbindung in den Problemraum, z. B. zu einem Feature-Modell, angelegt. Nachdem das Delta-Modell angelegt ist, wird das übertragene Instanz-Modell und ggf. auch das Instanz-Modell, auf dessen Grundlage das Delta-Modell erzeugt wurde, gelöscht.

Ergebnis: Das Delta-Modell und das Feature-Modell sind auf dem Server abgelegt und sind bereit zur Verwendung. Die Abhängigkeiten zwischen den beiden Modellen sind angelegt. Analog zu einer Versionsverwaltung für Quellcode ist dieser Ablauf der *Comitt* bzw. der *initial Comitt*.

Verteilen einer Lösung

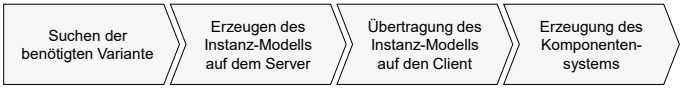


Abbildung 5.17: Ablauf zur Erzeugung eines Komponentensystems

Ausgangssituation: Auf einem Server liegen Delta- und Typ-Modelle vor. Die gleichen Typ-Modelle und die dazugehörigen Implementierung bzw. eine Referenz auf die Implementierungen sind auf dem Client vorhanden. Die entsprechenden Manager sind auf beiden Systemen vorhanden.

Ablauf: Im ersten Schritt wird vom Manager auf dem Client die benötigte Variante auf dem Server gesucht (vgl. Abbildung 5.17). Dies kann auf unterschiedliche Arten passieren: liegt nur eine Variantenbeschreibung im Lösungsraum (Delta-Modell) vor, muss die Variante allein über diese gefunden werden. Ist eine Abbildung im Lösungsraum (Feature-Modell) vorhanden, so kann diese durchsucht und das oder die Produkt(e) anhand der benötigten Features gefunden werden. Durch den Link auf das Delta-Modell, das dieses Produkt erzeugt, kann das Delta-Modell verwendet werden. Im zweiten Schritt wird das Instanz-Modell auf dem Server erzeugt. Dafür wird die Transformation *erzImpl* genutzt. Anschließend wird das erzeugte Instanz-Modell auf den Client übertragen und auf dem Server gelöscht. Unter Verwendung der Implementierungen auf dem Client wird aus dem vorliegenden Instanz-Modell die Implementierung erzeugt.

Ergebnis: Das durch ein Delta-Modell und seine verbundenen Modelle bis zum Root-Delta repräsentierte Komponentensystem sind in eine konkrete Implementierung umgewandelt. Diese liegt in der Umgebung vor und kann durch den Nutzer verwendet werden.

Erweitern einer bestehenden Lösung

Ausgangssituation: Ein Komponentensystem ist im Client vorhanden und das dazugehörige Delta-Modell liegt im Server vor. Die Typ-Modelle aller verwendeten Komponenten sind auf Server und Client vorhanden.

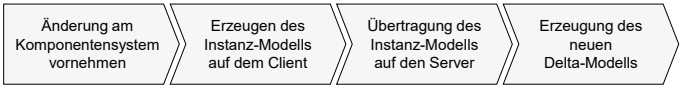


Abbildung 5.18: Ablauf zur Erweiterung einer bestehenden Lösung

Ablauf: Der vorgestellte Ablauf ist in Abbildung 5.18 dargestellt. Der erste Schritt ist die Änderung des Komponentensystems auf dem Client durch den Nutzer. Diese Änderung könnte ebenso direkt im Instanz-Modell erfolgen. Allerdings würde dieses Vorgehen das Look-and-Feel für den Nutzer ändern. Zusätzlich kommt hinzu, dass bei reinen Softwaresystemen die Änderungen am Komponentensystem automatisch in ein neues Instanz-Modell transformiert werden können (vgl. *erzKomponentenInstModell*), wodurch kein Mehraufwand im Vergleich zur Nutzung ohne das vorgestellte Konzept entsteht. Diese Umwandlung ist Schritt zwei des Ablaufs. Anschließend wird das Instanz-Modell auf den Server übertragen und auf dem Client gelöscht. Auf dem Server wird zwischen dem Instanz-Modell und dem Ausgangssystem das Delta-Modell erzeugt. Dieses wird mit dem Basisdelta und dem Feature-Modell verknüpft.

Ergebnis: Das geänderte Komponentensystem ist mit den Änderungen als Variante in den Varianten-Server aufgenommen worden und steht für die weitere Nutzung zur Verfügung. Die Verknüpfung zur Ausgangsvariante und die Einordnung in das Feature-Modell wurden vorgenommen.

Propagieren einer neuen Komponentenversion

Abbildung 5.19: Ablauf zum Propagieren einer neuen Version einer Komponente

Ausgangssituation: Es liegt eine verteilte Architektur vor und die neue Version einer Komponente wurde entwickelt. Die neue Komponente soll in einem Komponentensystem zum Einsatz kommen, das von einem Client verwaltet wird. Dies kann beispielsweise eine neue Version einer Pumpe sein oder eine verbesserte Version eines Funktionsbausteins.

Ablauf: Nach dem in Abbildung 5.19 dargestellten Ablauf ist der erste Schritt, das Typ-Modell zu identifizieren, das die neue Komponente realisiert. Im Modell wird eine Referenz auf die neue Version der Komponente eingetragen und als aktuelle Version markiert. Anschließend werden die von der Aktualisierung betroffenen Komponenten in den Komponentensystemen identifiziert. Dies können entweder alle Komponenten eines Typs sein, die von dem Client verwaltet werden, oder eine selektive Auswahl. Je nach Anwendungsfall kann es auch sein, dass die Aktualisierung nicht von dem verwaltenden Tool vorgenommen wird (*push*), sondern für jede Komponente einzeln angefordert wird (*pull*). Im nächsten Schritt wird die neue Version der Komponente instanziiert bzw. jedem Komponentensystem hinzugefügt. Ist die Beibehaltung des Namens nicht relevant, beispielsweise bei einer

Hardwarekomponente, die über ihre Rolle in der Anlage angesprochen wird, werden die Verbindungen der alten Komponente auf die neue übertragen. Wenn der Name beibehalten werden soll, muss in diesem Rahmen auch eine Umbenennung der neuen Komponente erfolgen. Nachdem die Verbindungen von der alten auf die neue Version übergegangen sind und die alte Komponente nicht mehr mit der Umgebung verbunden ist, kann sie aus dem Komponentensystem entfernt werden.

Ergebnis: Die neue Version einer Komponente ist in ein bzw. mehrere Komponentensystem(e) integriert. Die Systeme sind bereit zur anschließenden Verwendung.

Delta-Modell in anderem Kontext

Ausgangssituation: Es liegen ein Delta-Modell und ein Komponentensystem vor, auf das das Delta-Modell angewendet werden soll. Das Delta-Modell war ursprünglich für die Anwendung in einem anderen Kontext vorgesehen.

Ablauf: In einem ersten Schritt muss die Eignung des Delta-Modells für die Anwendung auf den neuen Kontext überprüft werden. Dies kann auf verschiedene Arten erfolgen. Eine erste Abschätzung kann anhand des Delta-Modells getroffen werden. Je mehr Verbindungen zu bestehenden Komponenten des Komponentensystems angelegt werden, desto höher ist die Wahrscheinlichkeit, dass eine von diesen nicht vorhanden ist. Ausgehend davon kann abgeschätzt werden, wie gut sich das Delta-Modell in die Umgebung integrieren lässt. Eine andere Möglichkeit ist es, Nutzen aus dem Komponenten-Modell zu ziehen und die Anwendung des Delta-Modells direkt zu testen. Anschließend muss durch Konsistenzprüfungen und aufgetretene Fehlermeldungen bei der Anwendung entschieden werden, ob diese erfolgreich war bzw. ob und wo nachgearbeitet werden muss. Ist der Prozess erfolgreich verlaufen, wird das (angepasste) Delta-Modell in den Delta-Server an der vorgesehenen Stelle integriert.

Ergebnis: Das Delta-Modell wurde für einen weiteren Anwendungsfall nutzbar gemacht und die in ihm enthaltenen Aufwände wurden erfolgreich wiederverwendet. Das Delta-Modell steht zur Nutzung in einem neuen Kontext zur Verfügung.

5.4 Kritische Betrachtung des Konzepts

Im folgenden Abschnitt wird das vorgestellte Konzept einer Betrachtung und Bewertung unterzogen. Zunächst werden die Vorteile des Konzepts herausgearbeitet. Anschließend werden Handlungsempfehlungen für die Verwendung gegeben. Abschließend werden die Randbedingungen für die Nutzung des Konzepts zusammengefasst.

5.4.1 Added Values

Ausgehend von Komponenten und Delta-Modellen in der Literatur wurden zwei Modelle entwickelt. Mit diesen können die Varianten von Komponentensystemen leicht und übersichtlich beschrieben werden. Die Nutzung der Komponenten-Modelle als Abstraktions-

schicht hat den Vorteil, dass so von der Implementierung der jeweiligen Komponente abstrahiert werden kann. Wie gezeigt, ist diese Vorgehensweise geeignet für die Übertragung der Komponentenstrukturen zwischen Systemen. Mit Hilfe des Delta-Modells kann die Variabilität im Lösungsraum intuitiv beschrieben werden, was somit eine Wiederverwendung von erzeugten Lösungen ermöglicht. Durch die Einfachheit und Nachvollziehbarkeit der Modelle wird eine höhere Akzeptanz bei den Mitarbeitern der unterschiedlichen Gewerke erreicht. Somit erhöht sich der Nutzungsgrad der Modelle. Im Konzept werden die Delta-Modelle auf ein Modell anstatt auf ein konkreten System angewendet. Da das Modell sehr allgemein formuliert ist, ist es sehr robust gegenüber Inkonsistenzen. So können Operationen und neue Anwendungen leicht erprobt und nachträglich deren Konsistenz sichergestellt werden.

Die verteilte Architektur hat den Vorteil, dass sie die Modelle in der Praxis anwendbar macht und gleichzeitig das Look-and-Feel der Nutzer möglichst wenig verändert. Die bewährten Prozesse und Tools können beibehalten werden. Lediglich für die Wiederverwendung wird, analog zu einem Versionsverwaltungstool für Code, ein zusätzliches Werkzeug benötigt. Darüber hinaus ermöglicht die Darstellung der Delta-Modelle in der domänenspezifischen Sprache die optimale Integration für den Nutzer und die vielfältige Verwendung der vorgestellten Modelle. Die verteilte Architektur vereint eine dezentrale Nutzung der Komponentensysteme mit der zentralen Speicherung der relevanten Informationen. So können diese jedem (potenziellen) Nutzer zugänglich gemacht werden.

Durch die zunehmende Verwendung von OPC UA und Laufzeitsystemen sinkt die Einstiegshürde für die Verwendung der vorgestellten objektorientierten Modelle, da diese nativ verwendet werden können. Gleichzeitig steigt der Bedarf der Verwaltung von komponentenbasierten Architekturen im Softwarebereich weiter an als er durch die Verwendung der IEC 61131 Sprachen schon gestiegen ist. Die Verwendung von Verwaltungsschalen und der darin enthaltenen Teilmodelle auf Basis von strukturierten Komponenten weist auf ein zusätzliches Anwendungsgebiet hin. Dieser Bedarf wird durch das vorgestellte Konzept gedeckt.

Zusätzlich wird durch das Konzept die explizite Modellierung von Komponentenversionen unterstützt. Von der Modellierung ausgehend, kann das beschriebene Verfahren für das Propagieren von neuen Versionen und weiterer Prozesse umgesetzt werden. In einem ersten Schritt ermöglicht die Modellierung, kompatible Komponentenversionen als solche darzustellen, und macht für den Nutzer kenntlich, wie diese in Komponentensystemen verwendet werden.

5.4.2 Randbedingungen

Im Folgenden werden die Randbedingungen und Einschränkungen, die bei der Verwendung des Konzepts zu berücksichtigen sind, zusammengefasst.

- Das Konzept ist besonders gut nutzbar, wenn die durch einzelne Delta-Modelle veränderte Funktionalität(en) orthogonal auf den Funktionalitäten des Komponentensystems stehen. Im Kontext von technischen Systemen ist die Trennung in orthogonale Funktionalitäten (z. B. aus [Har87]) bekannt. In der Automatisierungstechnik bietet es sich an, die Automaten der Steuerungskomponenten so zu entflechten, dass diese

orthogonal aufeinander stehen. Dementsprechend ist die Erfüllung dieser Randbedingung nicht mit großen Änderungen verbunden.

- Die Delta-Modelle sind immer nur für einen definierten Ausgangspunkt zu verwenden. Eine Anwendung auf eine andere Basis kann zu nicht definierten Ergebnissen bzw. zu einem inkonsistenten Komponentensystem führen. Ein einfaches Beispiel dafür ist das Löschen eines Objekts, das nicht existiert. In diesem Fall entspricht der Zustand dem angestrebten Ergebnis. Wird aber eine Verbindung zu einem Baustein angelegt, der nicht existiert, kann der gewünschte Zielzustand nicht erreicht werden. Das Komponenten-Modell und das Komponentensystem sind nach Anwendung der Delta-Operationen nicht in der gewünschten Weise aufgebaut.

- Eine unbeschränkte Menge von Delta-Operationen innerhalb eines Delta-Modells führt zu einer nicht vorhersagbaren Ausführungsdauer der Modifikation. Daher ist eine Verwendung zur Laufzeit ohne Einschränkung der Anzahl der Operationen nur bedingt möglich. Insgesamt kann das beschriebene Verfahren für einen Austausch bzw. den Aufbau von Komponentensystemen zur Laufzeit eingesetzt werden. Dies ist allerdings mit erhöhten Anforderungen an das Laufzeitsystem verbunden (z. B. Wechsel von Verbindungen und Umschalten zwischen Bausteinen zwischen zwei Zyklen). Zusätzlich muss die Funktionalität des stoßfreien Austausches von Baustein(system)en implementiert werden.

- Im vorgestellten Konzept wird das Delta-Modell hinsichtlich der strukturellen Differenz zwischen zwei Komponenten-Modellen gebildet. Dies bedeutet, dass Änderungen, die keine Auswirkungen auf die eigentliche Funktionalität haben, trotzdem in einem Delta-Modell abgebildet werden. Dies kann z. B. die Änderung des Namens einer Komponente sein. Je nach konkreter Umsetzung des Vergleiches kann die Namensgleichheit ein Kriterium für das Auffinden der Unterschiede sein.

- Durch die Verwendung eines einheitlichen Sets von Komponenten wird der Nutzen des Konzepts weiter gesteigert. Wie vorgestellt, lassen sich auch gewisse Abweichungen zwischen den Komponententypen nivellieren. Allerdings ist eine grundsätzliche Ähnlichkeit der Komponenten trotzdem erforderlich. Durch die Verwendung der IEC 61131 sind die Baustein-Bibliotheken in der Automatisierungstechnik zumindest funktional kompatibel. Ähnlich ist die Situation im Bereich der Hardwarekomponenten.

- Die Abgeschlossenheit der Komponenten muss diszipliniert vom Nutzer und von der Organisationseinheit durchgesetzt werden. Trotz der sinnvollen Forderung, diese sicherzustellen, sind globale Variable und direkte Lese- und Schreibzugriffe über die Grenzen von Komponenten hinweg weiterhin verbreitet. Ebenso muss eine einheitliche Richtlinie für Komponentennamen und die Pflege und Verwendung der Funktionalitätsreferenzen durchgängig angewendet werden.

5.4.3 Handlungsempfehlungen

Für jeden konkreten Anwendungsfall müssen eigene Regeln und Handlungsempfehlungen abgeleitet werden. Im Folgenden werden einige grundsätzliche Empfehlungen vorgestellt,

die die Nutzung des Konzepts einfacher machen.

Nach der Anwendung eines Delta-Modells sollte grundsätzlich die Konsistenz des erzeugten Komponenten-Modells überprüft werden. Grundlage dafür sind die Regeln des abgebildeten Komponentensystems. Dies können unter anderem die Umwandlungsregeln der jeweiligen Implementierungen sein. Demzufolge muss die Konsistenzprüfung auf dem Client unter Berücksichtigung der lokal geltenden Regeln erneut durchgeführt werden. Alternativ wird die Prüfung anhand von unterschiedlichen Profilen je nach verwendetem Client durchgeführt. Die Unterschiede durch die Implementierungen sind durch die Verwendung von gemeinsamen Sprachen gering.

Zusätzlich können auf dem Instanz-Modell Überprüfungen durchgeführt werden, die nicht direkt eine Inkonsistenz anzeigen, dennoch einen Indikator für eine nicht optimale Anwendung eines Delta-Modells darstellen. Insbesondere nach der Anwendung eines Delta-Modells in einem neuen Kontext ist eine solche Überprüfung sinnvoll. In einem ersten Schritt ist zu prüfen, ob die Eingangsports des Systems offen, d. h. nicht mit mindestens einer Komponente des Systems verbunden sind. Ebenso sollen die Ausgangsports des Gesamtsystems nicht un-parametriert bzw. unverbunden sein. Beide Zustände können Indikatoren für eine fehlerhafte Integration oder Artefakte von anderen Varianten sein. Eine weitere mögliche Erklärung ist ein Informationsfluss über die Grenzen der Komponente hinweg ohne das Nutzen der dafür vorgesehenen Ports. Dies ist eigentlich durch die Kapselung von Komponenten unterbunden, kommt in der Praxis allerdings vor. Hat eine Komponente keine oder nur eine einseitige Verbindung zu ihrer Umgebung, so kann dies ebenso ein Indikator für eine fehlerhafte Integration sein. Dass beispielsweise eine Pumpe nur an einer Seite mit einem Rohr verbunden ist, weist in der Mehrzahl der Fälle auf einen Fehler hin.

Je nach Anforderungen der Nutzer kann es sinnvoll sein, die Delta-Modelle nach Gesichtspunkten der Komplexität oder der Funktionalität aufzubauen. Funktionalität bedeutet hier, dass ein Delta-Modell mit einer oder mehreren abgeschlossenen Funktionalitäten assoziiert ist. Darüber hinaus kann es sinnvoll sein, Delta-Modelle in mehrere weniger komplexe Delta-Modelle zu zerteilen. In einem anderen Anwendungsfall kann die Zusammenfassung zu einem größeren Delta-Modell passender sein. So kann der Baum aus Delta-Modellen je nach den lokalen Gegebenheiten aufgebaut werden. Nur, wenn solche Regeln aufgestellt und eingehalten werden, ist die Nutzung eines Delta-Servers als „gelbe Seiten“ für Lösungen denkbar und ein wirklicher Fortschritt im Hinblick auf die Wiederverwendbarkeit.

6 Prototypische Realisierung und Anwendungsfälle

Im folgenden Kapitel wird die prototypische Umsetzung des vorgestellten Konzepts beschrieben. Die Nutzung dieser Implementierung wird anschließend an drei Anwendungsfällen verdeutlicht. Abschließend wird am Ende des Kapitels die Implementierung evaluiert. Dies geschieht anhand von ausgesuchten Szenarien, die die Leistungsfähigkeit und Reaktion der Implementierung auf Fehler verdeutlichen.

6.1 Implementierung in ACPLT/RTE

Die Umsetzung des Konzepts erfolgte in der Laufzeitumgebung des Lehrstuhls für Prozessleittechnik ACPLT/RTE [GE13]. Der Laufzeitumgebung liegt ein Meta-Metamodell für Objekte zugrunde, auf dem alle weiteren Implementierungen aufsetzen (vgl. Kapitel 4.4.1). Auf diesem Meta-Metamodell aufbauend wurde eine Funktionsbausteinarchitektur realisiert, mit der es möglich ist, IEC 61131 konforme Lösungen umzusetzen. Unter anderem wurde so die Prozessführung für Anlagen implementiert [WTPE17, WE15a]. Darüber hinaus bietet die Laufzeitumgebung die Möglichkeit, Modelle z. B. für die Beschreibung von Anlagen zu implementieren und zu erproben [YGE13, Sch16a]. Durch diese Modelle und geeignete Modelltransformationen können Automatisierungslösungen synthetisiert werden [WKS+16].

Damit die Laufzeitumgebung auf unterschiedlicher Hardware ausgeführt werden kann, ist sie in ANSI C implementiert. Sie realisiert ein Konzept von zur Laufzeit nachladbaren Bibliotheken. So können Funktionalitäten in Bibliotheken implementiert und zur Laufzeit nachgeladen werden, damit anschießend die Objektstruktur konfiguriert werden kann.

Die Anbindung der Laufzeitumgebung an andere Anwendungen erfolgt über das Kommunikationsprotokoll des Lehrstuhls ACPLT/KS [WE17] und über OPC UA [GPP16]. Als Benutzerschnittstelle stehen verschiedene Engineeringtools zur Verfügung, die auf die jeweiligen Anwendungsfälle zugeschnitten sind. Das bedeutet, dass einige Tools universell für die Konfiguration der Laufzeitumgebung und der enthaltenen Objektstrukturen einsetzbar sind. Andere sind spezifisch für die Interaktion mit Objekten einer Bibliothek entwickelt worden (z. B. Dienste-Klienten).

Analog zum Konzept besteht die Struktur der Implementierung aus zwei Teilen. Dabei handelt es sich um die Modelle und den Manager. Diese müssen für die Nutzung reibungslos zusammenspielen. Im Folgenden werden sie zur besseren Verständlichkeit nacheinander vorgestellt.

6.1.1 Umsetzung der Modelle

Die Modelle wurden entsprechend der in Kapitel 5 vorgestellten Spezifikationen in der Laufzeitumgebung ACPLT/RTE umgesetzt. Die Klassen der Komponenten-Metamodelle und des Delta-Modells wurden in Klassen der Laufzeitumgebung umgesetzt. Entsprechend der Metamodelle sind die Klassen in zwei Bibliotheken strukturiert. Zusätzlich zu den in den Metamodellen beschriebenen Klassen sind in den Bibliotheken Klassen für Managerobjekte enthalten. Die Managerobjekte beinhalten Operationen zur Realisierung der vorgestellten Transformationen.

Der *componentClassManager* realisiert die Verwaltung der Typ-Modelle der Komponenten. Der Manager beinhaltet die Funktionalität zur automatischen Erzeugung des Typ-Modells einer Bibliothek der Laufzeitumgebung. Als alternative Quelle zur Erzeugung des Typ-Modells nutzt der Manager einen Prototypen des Komponentensystems. Er besitzt einen Port, in dem der Pfad zu den Bibliotheksobjekten in der Laufzeitumgebung oder dem Prototypen angegeben wird. Ausgehend davon wird die Bibliothek erkundet und das Modell aufgebaut. Der Aufbau des Modells wird durch die drei Operationen der Klasse *componentClass* realisiert. Eine Operation ist die Erzeugung des Typ-Modells aus einer Funktionsblockklasse. Die zweite Operation nutzt ein Komponentensystem als Prototyp und baut daraus das Typ-Modell auf. Die dritte Operation der Klasse ist die Erzeugung des Instanz-Modells aus dem jeweiligen Typ-Modell.

Das *componentChart* ist eine Klasse, die den Rahmen eines Instanz-Modells mit dem zugehörigen externen Interface bildet. Sie verfügt über eine Operation zum Erzeugen des Instanz-Modells aus einem Komponentensystem und eine Operation zum Umwandeln des Instanz-Modells in die dazugehörige Implementierung. Zur Verwaltung eines oder mehrerer *componentCharts* sind *componentChartManager* vorgesehen. Der Manager erzeugt Instanz-Modelle aus Implementierungen oder Implementierungen aus Instanz-Modellen durch Orchestrierung der Operationen der *componentCharts*. Dabei stellt der Manager die Eindeutigkeit von Komponentennamen sicher und verwaltet die Ziel- und Quellpfade.

Geschachtelte Komponentensysteme haben grundsätzlich eine beliebige Tiefe. Das vorgestellte Konzept lässt es zu, die Komponentensysteme bis zur letzten Ebene in ein Modell zu überführen. Im Rahmen der Implementierung wird eine beliebige Modelltiefe angenommen. Allerdings ist die Erkundung von bekannten Strukturen ausschließlich auf die ersten Ebene beschränkt. Geschachtelte Strukturen werden als eingebettet bzw. nested bezeichnet und durch Prototypen abgebildet. Diese Prototypen können selbst Varianten voneinander sein. Die entsprechenden Delta-Modelle können in Typ-Modellen als Implementierung verwendet werden. Der *componentChartManager* erkennt beim Umwandeln eines Komponentensystems anhand des Objekttyps, ob es sich um eine eingebettete Unterstruktur handelt. In diesem Fall wird anhand des Links zum Typ-Modell geprüft, ob diese Struktur bereits als Typ-Modell vorliegt. Liegt die Struktur vor, wird diese im Modell verlinkt. Andernfalls wird sie in einem neuen Typ-Modell nachgebildet. Dieser Prozess kann durch alle Hierarchieebenen durchlaufen werden. Somit stellt die Einschränkung auf eine Ebene keine Beschränkung der Allgemeinheit dar. Der Vergleichsmechanismus für die eingebundenen Strukturen kann so weiterentwickelt werden, dass die Nutzung von Delta-Modellen für den Vergleich zweier Implementierungen ebenfalls denkbar ist. Wenn das Delta-Modell leer ist, sind die beiden verglichenen Instanz-Modelle gleich. Im Rahmen dieser Implementierung

wurde darauf verzichtet, da die Aussagekraft im Hinblick auf das Konzept gering ist.

Die Implementierung des Delta-Modells erfolgt auf der untersten Ebene durch die Realisierung der in Abbildung 5.6 dargestellten Operationen. Dies wird durch die entsprechenden Objekte und ihre Methoden in der Bibliothek *dvariantsDelta* realisiert. Diese Operationen werden jeweils in einem *Delta* zusammengefasst. Dieses bietet die Methoden an, alle Delta-Operationen, die in ihm enthalten sind, auf ein gegebenes *componentChart* anzuwenden. Die Operation *createComponentChart* erzeugt das durch dieses konkrete Delta-Modell dargestellte Komponentensystem. Dies wird durch die rekursive Ausführung aller Delta-Modelle ausgehend vom jeweiligen Root-Delta erreicht. Umgekehrt wird ein Delta-Modell durch die Operation *createFromComponentCharts* aus zwei vorliegenden Instanz-Modellen (*componentCharts*) generiert.

Die Delta-Modelle werden durch die *deltaManager* verwaltet und durch die angebotenen Funktionen besser zugänglich gemacht. Durch das Setzen der jeweiligen Parameter wird z. B. die Basis-Variante eines zu erzeugenden Deltas festgelegt.

Das Feature-Modell wurde, wie in Abbildung 5.11 dargestellt, umgesetzt. Da das Modell in diesem Kontext eine rein deskriptive Rolle einnimmt, war die Implementierung von Funktionalitäten innerhalb des Modells nicht nötig. Die Verbindungen mit den Delta-Modellen werden über das Server-Objekt angelegt (vgl. Abschnitt 6.1.2).

Für die Umsetzung der Visualisierung wurde ein zusätzliches Modell innerhalb der Delta-Modell-Bibliothek angelegt. Diese ermöglicht es, den Informationsgehalt eines Delta-Modells und des korrespondierenden Instanz-Modells in einem Modell darzustellen. Es ist aufgebaut wie ein Instanz-Modell, allerdings gibt eine zusätzliche Variable an, ob eine Instanz hinzugefügt, gelöscht oder bearbeitet wurde.

6.1.2 Realisierung der dezentralen Struktur

Die verteilte Struktur wird durch ein Server- und ein Client-Objekt realisiert. Zusätzlich steht ein *deltaExplorer* genannter Baustein zur Verfügung, der es ermöglicht, einen Delta-Server zu erkunden und nach den benötigten Varianten zu durchsuchen. Die Suche erfolgt anhand des Feature-Modells und liefert als Ergebnis eine Menge von zu den Suchkriterien kompatiblen Delta-Modellen.

Als Kommunikationsmittel kommen auf der Seite des Clients und des Explorers *ks-Bausteine* zum Einsatz. Mit diesen werden Informationen und Aufträge in die entsprechenden Eingänge des Server-Objekts geschrieben und die Ergebnisse aus den Ausgängen ausgelesen. Der Austausch erfolgt über die Serialisierung der Instanz-Modelle (*componentCharts*) und die hinzugefügten Aufrufe. Vorteilhaft an dieser Umsetzung ist, dass die verwendete JSON Serialisierung sehr kompakt, durch Menschen interpretierbar und weit verbreitet ist. Somit stehen d. h. Parser für die meisten gängigen Programmiersprachen zur Verfügung. Eine Änderung des Kommunikationsprotokolls auf z. B. OPC UA ist durch den modularen Aufbau einfach möglich. Dazu muss der entsprechende Baustein zum Lesen und Schreiben ausgetauscht werden.

Über die Ein- und Ausgänge des *deltaClients* werden der Zielserver und der internen Zustands des Bausteins angegeben. Die Hauptfunktionen sind das *checkout* und das *commit*.

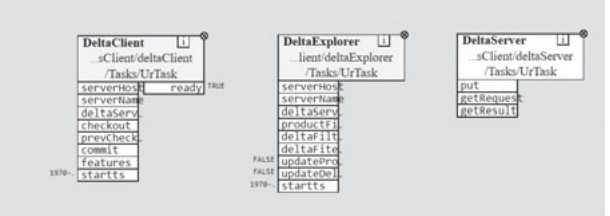


Abbildung 6.1: Server, Client und Explorer Bausteine im HMI.

Mit dem checkout wird ein bestehendes Delta-Modell vom Server auf den Client heruntergeladen. Mit der Funktionalität commit werden Implementierungen auf dem Client in ein Delta-Modell auf dem Server umgewandelt und dort in den Baum der Delta-Modelle eingefügt.

Das Gegenstück dazu auf dem Server ist ein Objekt der Klasse *deltaServer*. Dieses stellt den Zugangspunkt für die Clients dar und gibt die eingehenden Anfragen an die Komponenten-, Delta- und Feature-Modelle weiter. Das Objekt stellt die Funktionalitäten *commmit*, *checkout* und *explore* für das zugrunde liegende Delta-Modell bereit. So können die Funktionen vom Client verwendet werden. Die vorgestellten Client-, Server- und Explorer-Bausteine sind in Abbildung 6.1 dargestellt.

In Abbildung 6.2 ist der Ablauf eines *checkouts* mit den daran beteiligten Objekten als UML-Sequenzdiagramm dargestellt. Auf der linken Seite der Abbildung ist der Nutzer des Systems zu erkennen. Dabei kann es sich um einen Menschen oder eine andere Anwendung handeln. Die weiteren dargestellten Objekte werden danach unterteilt, ob sie sich auf dem Client (*deltaClient* und Komponenten-Modell) oder auf dem Server befinden (*deltaServer*, Delta-Modell und Komponenten-Modell). Zu Beginn wird der *deltaClient* vom Nutzer parametriert. Bei den eingestellten Parametern handelt es sich um die Adresse des Servers und des Delta-Modells. Anschließend wird der *checkout* unter Angabe des herunterzuladenden Komponentensystems durch den Nutzer ausgelöst. Der Delta-Client übermittelt die Anfrage unter Verwendung eines JSON Strings an den *deltaServer*. Dieser sucht das angeforderte Delta-Modell und baut es, wenn vorhanden, zusammen. Für den Zusammenbau wird ermittelt, ob ein Basis-Delta zu dem Delta-Modell existiert. Ist das nicht der Fall, werden die Delta-Operationen unter Einbeziehung des Typ-Modells angewendet und das Instanz-Modell aufgebaut. Sollte ein Basis-Delta vorliegen, wird die Operation rekursiv auf die Basis-Deltas angewendet bis das Root-Delta erreicht ist. Anschließend werden diese der Reihe nach ausgeführt und das kumulierte Instanz-Modell erzeugt. Dieses wird dem *deltaServer* zur Verfügung gestellt, der es serialisiert und an den *deltaClient* übermittelt. Dort wird es deserialisiert und unter Einbeziehung des lokalen Typ-Modells wird aus dem empfangenen Instanz-Modell das Komponentensystem erzeugt. Der erfolgreiche Abschluss des Vorgangs oder eine Fehlermeldung wird dem Nutzer über einen Statusausgang mitgeteilt.

Analog zum checkout ist der Ablauf des *commits* in Abbildung 6.3 dargestellt. Dieser startet ebenso mit dem Parametrieren des *deltaClients* durch den Nutzer. Anschließend wird unter Angabe des zu übertragenden Komponentensystems, ggf. des Basis-Deltas und der Features der Prozess gestartet. Unter Einbeziehung des Komponenten-Modells wird aus

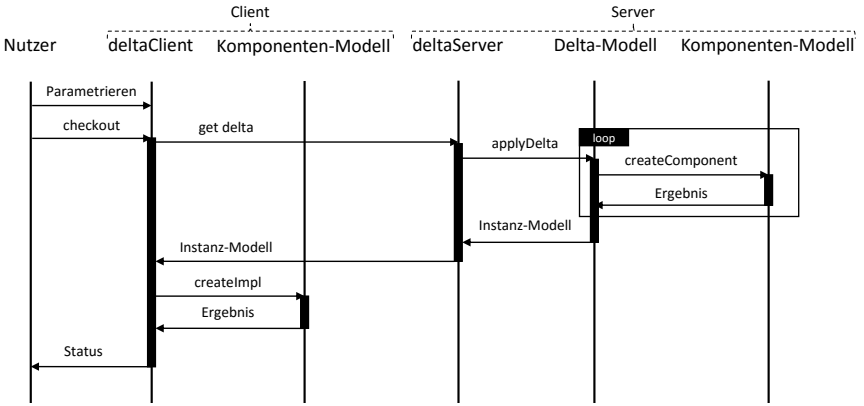


Abbildung 6.2: Sequenzdiagramm zur Beschreibung des Checkouts eines Komponentensystems vom Server auf einen Client.

dem Komponentensystem das Instanz-Modell erzeugt. Dieses wird durch den *deltaClient* serialisiert und an den Server geschickt. Dieser erzeugt daraus das Instanz-Modell und, falls benötigt, das Instanz-Modell des Basis-Deltas. Dieses wird unter Einbeziehung des Komponenten-Modells rekursiv aufgebaut. Wenn beide Instanz-Modelle vorliegen, wird das Delta-Modell zwischen beiden gebildet. Das Basis-Delta kann dabei allerdings auch leer sein. Das Delta-Modell wird anschließend in den Delta-Baum an der entsprechenden Stelle eingefügt und das Feature-Modell um die entsprechenden Einträge ergänzt. Dem Nutzer wird danach der erfolgreiche Abschluss des Vorgangs oder eine Fehlermeldung mitgeteilt.

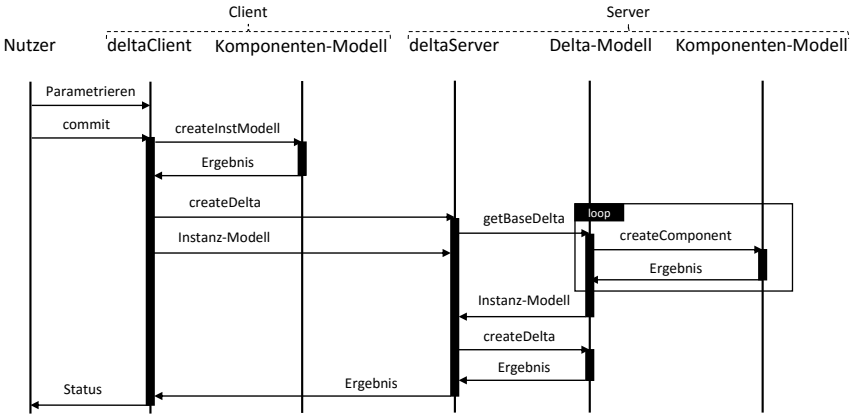


Abbildung 6.3: Sequenzdiagramm zur Beschreibung des Commits eines Komponentensystems vom Client zu einem Server.

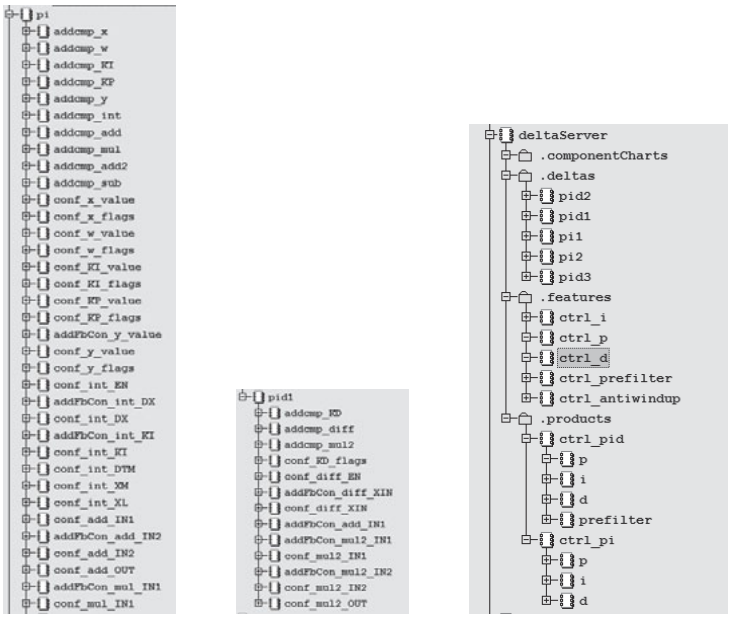
6.2 Anwendungsfälle

Im folgenden Abschnitt werden drei Anwendungsfälle für das Konzept und die Implementierung vorgestellt. Diese verdeutlichen exemplarisch den Nutzen des Konzepts. Das erste Beispiel ist der Baustein für die Umsetzung von PID-Reglern. Anschließend werdend die am Lehrstuhl für Prozesstechnik entwickelten und verwendeten Prozessführungskomponenten als Beispiel vorgestellt. Abschließend wird am Beispiel der modularen Anlage M4P.AC ein hybrides System betrachtet. Es ist zu berücksichtigen, dass der Aufbau eines Delta-Baums grundsätzlich nicht eindeutig ist. Für die Nutzung bedeutet dieser Umstand, dass der Aufbau der Modelle eine Wahlmöglichkeit des Anwenders ist. Durch die Möglichkeiten der Transformation kann ohne Einschränkungen zwischen den Varianten gewechselt werden. In der Darstellung der Anwendungsfälle liegt der Fokus auf den Delta-Modellen und den Vorteilen, die sich aus der Verwendung des Konzepts ergeben.

6.2.1 PID-Regler-Baustein

Die Gestaltung eines PID-Bausteins in verschiedenen Leitsystemen unterscheidet sich im Hinblick auf den Funktionsumfang, die Art der Implementierung und die Nutzung. In der Vergangenheit gab es z. B. in der VDI/VDE 3696 [VDI95] den Vorschlag für einen Standard-PID-Regler. Da sich dieser in aktuellen Implementierungen nicht durchgesetzt hat, wurde im Rahmen des NAMUR AK 2.2 erneut der Versuch unternommen, einen Standard-Baustein zu spezifizieren. Da der benötigte Funktionsumfang sehr subjektiv ist und jeder Anwender und jedes Unternehmen eigene Anwendungsszenarien vor Augen hat, konnte sich der Arbeitskreis nur auf zwei Varianten des Reglers einigen. Aus diesem Grund stellt der Regler ein sehr gutes Beisiel für die Nutzung des vorgestellten Konzepts dar. Die Forderung, dass der Baustein ein einheitliches Interface und Verhalten aufweist, kann

durch das Konzept erfüllt werden. Gleichzeitig kann die Funktionalität flexibel auf die Wünsche und Erwartungen des Nutzers zugeschnitten werden.



(a) PI-Regler **(b)** PID-Regler **(c)** Server-Manager mit Delta- und Feature-Modellen.

Abbildung 6.4: Delta-Modelle und Aufbau des Server-Managers am Beispiel von PID-Reglern.

Im Folgenden wird der Aufbau von PID-Varianten aus Komponentensystemen betrachtet. Grundlage dafür sind Bausteine, die in der IEC 61131 spezifiziert sind. Eine Übersicht der Varianten des PID-Reglers aus [WE15b] ist in Abbildung 4.5 dargestellt. Exemplarisch wurde das Typ-Modell der benötigten Funktionsbausteinbibliotheken erzeugt und ausgehend davon die Delta-Modelle zwischen den Varianten gebildet. In Abbildung 6.4 sind beispielhaft drei Delta-Modelle in der Ansicht eines Engineeringsystems dargestellt. Ein Ausschnitt aus dem Root-Delta ist in Abbildung 6.4a zu erkennen. Ein Delta-Modell für die Fortentwicklung zum PID-Regler ist in Abbildung 6.4b zu erkennen. Der Ausbau eines Delta-Servers mit den Delta- und Feature-Modellen ist in Abbildung 6.4c zu sehen.

6.2.2 Prozessführungskomponenten

Die in Abschnitt 2.1 eingeführten Prozessführungskomponenten bestehen aus Teilsystemen, die in vielen Anwendungen verwendet werden können. Der Unterschied der einzelnen Prozessführungskomponententypen zueinander ist teilweise sehr gering. Allerdings ist er so groß, dass eine Nutzung der gleichen Komponente nicht möglich ist. Zusätzlich beinhalten

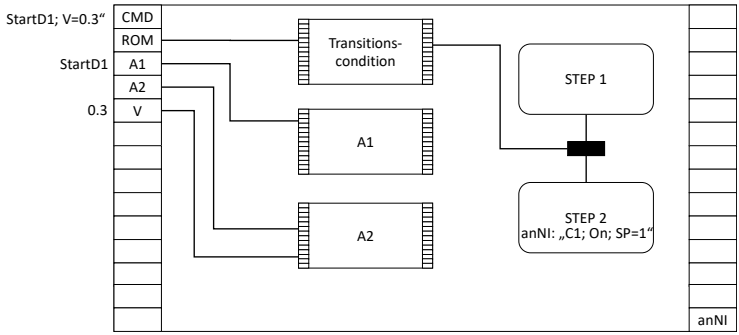


Abbildung 6.5: Prinzipieller Aufbau einer Prozessführungskomponente [WE15a]

die Komponenten orthogonal zueinander implementierbare Funktionen. Durch den geringen Abstand und die große Variantenvielfalt sind Prozessführungskomponenten ein gutes Anwendungsgebiet für das vorgestellte Konzept. Bereits bei Anwendungsfällen mit einer geringen Variantenvielfalt und wenigen Funktionalitäten steigt der Implementierungs- und Wartungsaufwand ohne einen Mechanismus zur Wiederverwendung sehr stark an.

Der prinzipielle Aufbau einer Prozessführungskomponente ist in Abbildung 6.5 zu erkennen. Die Prozessführungskomponente besteht aus dem Ausführungsrahmen und den enthaltenen Komponenten. Der Rahmen stellt die Schnittstelle zur Umgebung der Komponente dar und beinhaltet die Ausführungslogik. Am oberen linken Rand ist der Eingang für die Kommandos zur Prozessführung dargestellt. Die Kommandos werden durch den Ausführungsrahmen an den Eingang des Automaten, der dieses Kommando verarbeitet, zugestellt. Beispielhaft ist dargestellt, dass das Kommando *StartD1* an den Eingang des Automaten *A1* übermittelt wurde. Eine Schrittkette kann ebenso ohne weitere Kapselung in der Prozessführungskomponente realisiert werden (*Step 1* und *Step 2*). In Step 2 ist ein Kommando zu erkennen, das über den Ausgang anNI an eine weitere Komponente übermittelt wird. Durch die Kommandos werden die angebotenen Dienste der Prozessführungskomponenten aufgerufen [WTPE17].

Ein Baustein zum Erkunden und Aufrufen der angebotenen Dienste einer Prozessführungskomponente ist in Abbildung 6.6 dargestellt [WE17]. Der Baustein modifiziert seine eigenen Schnittstellen so, dass sie den Parametern der Prozessführungskomponente gleichen. Dies wird über die Erkundung der Prozessführungskomponente umgesetzt. Dafür verwendet der Baustein die Schnittstelle zu einem Dienstsystem. Diese Schnittstelle wird in der Implementierung durch zusätzliche Bausteine realisiert. Wenn ein Kommando im Eingang *CMD* vorliegt und der Eingang *SEND* auf TRUE gesetzt wird, wird der Dienstaufruf einschließlich der Werte für die Eingangsparameter an die Prozessführungskomponente übertragen. Die Ausgangsparameter des Bausteins werden regelmäßig über Abfragen aktuell gehalten. Der Baustein ermöglicht es, aus einer IEC 61131 Umgebung über ein Dienstsystem auf einen Diensterbringer zuzugreifen. Dafür ist der Baustein flexibel umgesetzt, um mit unterschiedlichen Diensterbringern umgehen zu können.

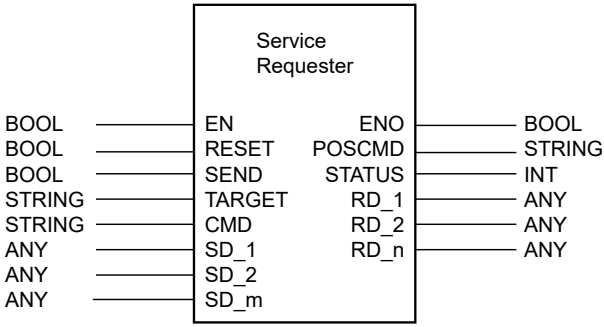


Abbildung 6.6: Schnittstelle eines Bausteins zum Dienstaufruf [WE17]

In Abbildung 6.7 ist der Aufbau des Basys-Demonstrators zu erkennen. Es handelt sich dabei um die Simulation eines Transportsystems für Paletten, auf denen aufgewickelte Metallbänder (Coils) transportiert werden.

Die Rechtecke in der Darstellung mit der Bezeichnung PEXX stellen Rollgänge dar. In diesen befinden sich die Antriebseinheiten, um die Paletten voran zu treiben. In diesem Beispiel existieren vier Typen von Rollgängen:

- Normal: Bidirektionaler Transport in horizontaler (Bild-)Richtung.
- Verfahrgänge: Bidirektionaler Transport in horizontaler und vertikaler (Bild-)Richtung.
- Drehteller: Bidirektionaler Transport in horizontaler (Bild-)Richtung und Drehen des Rollgangs um 180°.
- Ofen: Bidirektionaler Transport in horizontaler (Bild-)Richtung und Erhitzen der Ladung nach dem Schließen der Tore.

An der dargestellten Liste ist zu erkennen, dass die Funktionalität der verwendeten Typen in einigen Punkte deckungsgleich ist. So benötigen alle Rollgänge die Funktionalität des bidirektionalen Transports. Die weiteren Funktionalitäten können additiv hinzugefügt werden. Es wurden alle Rollgänge als Varianten des normalen Rollgangs mit erweiterten Funktionalitäten betrachtet. Davon ausgehend wurden die Funktionalitäten implementiert und die Transformationen in Form von Delta-Modellen abgelegt. In Abbildung 6.8 sind exemplarisch zwei dieser Delta-Modelle dargestellt. Auf der linken Seite (Abbildung 6.8a) ist ein Auszug aus dem Root-Delta für den normalen Rollgang zu erkennen. Das Delta-Modell für die Integration der Funktionalität *Drehteller* in den Prozessführungsbaustein ist auf der rechten Seite (Abbildung 6.8a) dargestellt.

Ein Vorteil dieser Herangehensweise gegenüber der vorher gängigen Praxis des Copy-and-Modify ist, dass eine Änderung an einem Komponentensystem in alle von diesem abgeleiteten Komponentensysteme eingefügt wird. Bei nachträglicher Änderungen an einem Delta-Modell bzw. einem Komponentensystem ist zu prüfen, ob es sich um das Beheben

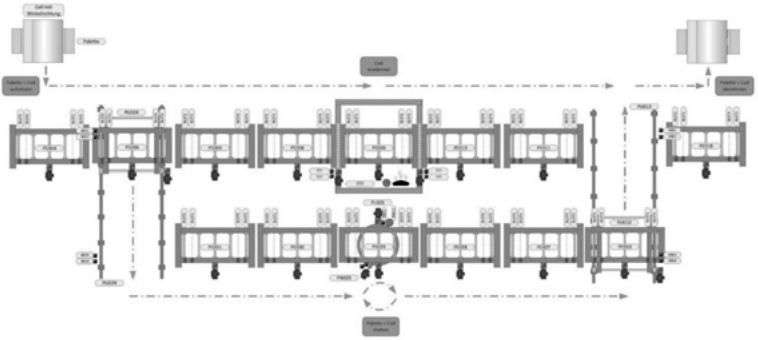
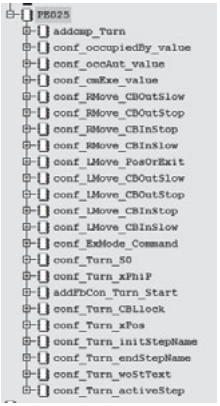
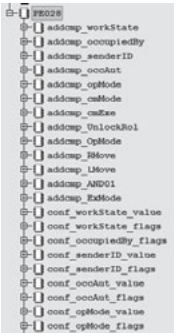


Abbildung 6.7: Übersicht über den Aufbau des verwendeten BaSys-Demonstrators.

eines Fehlers oder eine neue Variante handelt. Handelt es sich um eine neue Variante, ist das Komponentensystem vom Ausgangssystem abgeleitet und es muss ein neues Delta-Modell erstellt werden. Zusätzlich bietet die Herangehensweise die Möglichkeit, die Implementierungen in anderen Anwendungsfällen und Projekten wiederzuverwenden, da ein Austausch oder eine Ergänzung von Funktionalitäten leicht möglich ist. Durch die Verwendung der implementierten Transformationen ist der Mehraufwand im Rahmen des Engineerings überschaubar und insbesondere bei Fehlerbehebungen stellt das neue Vorgehen eine deutliche Verbesserung im Hinblick auf den Zeitaufwand dar.



(a) Ausssschnitt Root-Delta



(b) Drehteller

Abbildung 6.8: Delta-Modelle für die Beschreibung von Prozessführungsbausteinen.

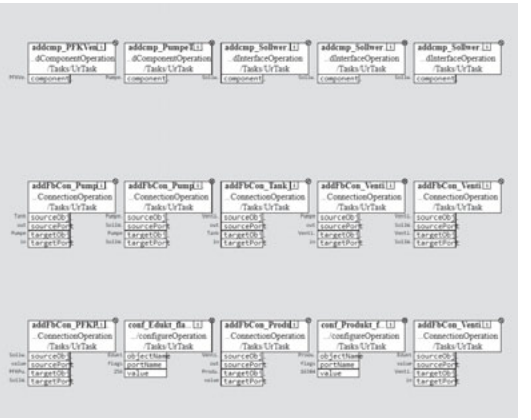
6.2.3 Modulare Anlage M4P.AC

Der im Folgenden beschriebene Anwendungsfall befasst sich mit der Anwendung des vorgestellten Konzepts auf ein hybrides System. Da es sich um ein hybrides System handelt und die eingeführte Implementierung den Zusammenbau eines Hardwaresystems nicht unterstützt, ist der Anwendungsfall im rein deskriptiven Bereich angesiedelt. Trotz dieser Einschränkung ist der Anwendungsfall für die Automatisierungstechnik relevant, da jedes System einen Hardwareteil beinhaltet. Die Umwandlung in das konkrete Komponentensystem benötigt in diesem Fall die Unterstützung eines Menschen bzw. einen vollständig automatisierten Fertigungsprozess.

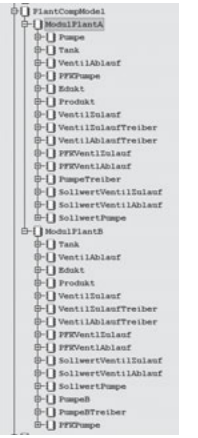
Beispielhaft für ein hybrides System ist die modulare Anlage M4P.Ac, die am Lehrstuhl für Prozessleittechnik entwickelt und betrieben wird. Sie besteht aus drei Waben, die je nach Produktionsauftrag und produziertem Produkt auf unterschiedliche Weise zusammengeschlossen werden können. Jede Wabe verfügt über einen anderen technischen Aufbau. Dies bedeutet eine unterschiedliche Hardware und Automatisierungslösung.

Der beispielhafte Aufbau eines Delta-Baums für die Beschreibung solcher Module ist in Abbildung 5.9 zu finden. Vorteilhaft bei dieser Herangehensweise ist die Nutzung von Lösungen über die Grenzen von Herstellern hinweg. Wenn eine Hard- oder Softwarekomponente ausgetauscht wird, hat dies unter Umständen Auswirkungen auf die umgebenden Komponenten. Diese Auswirkungen können von einem erneuten Parametrieren bis zu einem Austausch (von z. B. einem Treiberbaustein) reichen. Es ist wichtig zu berücksichtigen, dass die Delta Varianten und die Betrachtung als Komponentensystem auf verschiedenen Hierarchieebenen möglich ist. So können die unterschiedlichen Varianten eines Moduls durch die Änderung einer Variante eines Tanks oder einer Pumpe sowie des dazugehörigen Steuerungssystems und der dazugehörigen Steuerung erzeugt werden. Durch die Anwendung des Konzepts wird das Modul mit seinen Komponenten beschrieben und ebenso die Programmierung der Steuerungen als eine dieser Komponenten. Insbesondere bei den Steuerungen erfolgt das Übersetzen der Komponenten-Modelle in die konkreten Implementierungen je nach System möglicherweise anders. Dies zeigt die vielfältigen Freiheitsgrade bei der Anwendung des Konzepts.

Für diesen Anwendungsfall wurde eine Bibliothek erstellt, die die Funktionalität der Komponenten dieses Systems in der Laufzeitumgebung nachbildet. Aus dieser wurden die Typ-Modelle generiert. Aus den simulierten Komponentensystemen konnten anschließend die Instanz-Modelle erzeugt werden. Beispielhaft ist ein Instanz-Modell in Abbildung 6.9b zu erkennen. In Abbildung 6.9a ist ein Delta zur Beschreibung der modularen Anlage dargestellt.



(a) Root-Delta der modularen Anlage mit den Operationen.



(b) Instanz-Modell der modularen Analge

Abbildung 6.9: Delta-Modelle für die Beschreibung von Prozessführungsbausteinen.

6.3 Evaluierung der Implementierung

Im Folgenden wird zur Evaluierung der Funktionsfähigkeit der Implementierung eine Reihe von exemplarischen Anwendungsszenarien vorgestellt. Zu jedem Anwendungsszenario wird die Reaktion der Applikation beschrieben. Die Übersicht der Szenarien ist in Tabelle 6.1 zu finden. Es ist zu erkennen, dass die aufgeführten Szenarien aus Normal- und Fehlerfällen bestehen. Die Reaktion auf einen detektierten Fehler ist Anwendungs-spezifisch. So ist das Löschen eines Objekts durch die Anwendung eines Delta-Modells, das nicht existiert, ein Fehler. Allerdings beeinträchtigt dieser Fehler nicht die Funktionalität der Implementierung. Gleichwohl ist er ein Indikator, dass die Anwendung des Delta-Modells nicht wie vorgesehen funktioniert hat. Dies könnte zum Beispiel die Anwendung eines Delta-Modells in einem neuen Kontext sein. Ziel der Implementierung ist in diesem Fall, den Nutzer auf den Fehler aufmerksam zu machen und mit dem Prozess fortzufahren.

Tabelle 6.1: Übersicht über die Anwendungsszenarien

Fallbeschreibung	Ergebnis der Applikation
Modellerzeugung (Softwaresysteme)	
Erzeugen eines Typ-Modells aus einer Bibliothek	Anlegen des Modells
Erzeugen des Instanz-Modells aus dem konkreten System	Anlegen des Modells
Erzeugen des Instanz-Modells aus dem konkreten System bei fehlenden Typ-Modellen (Blackbox Komponenten)	Partielles Anlegen des Modell und Ausgabe der Fehlenden Elemente, die nicht verknüpfbaren Verbindungen als Konsequenz werden ebenso gemeldet
Erzeugen des Instanz-Modells aus dem konkreten System bei fehlenden Typ-Modellen (Whitebox-Komponenten)	Anlagen des entsprechenden Modells, Ausgabe einer Meldung, hinzufügen der Whitebox-Komponente zu den Typ-Modellen
Erzeugen eines Delta-Modells aus zwei Komponentensystemen	Anlegen des Delta-Modells verbunden mit der Einordnung in ein Feature-Modell
Anwendung der Delta-Modelle	
Anwenden einer korrekten Hinzufüge-Operation (Komponenten-Instanz, Interface-Instanz)	Ausführung der Operation
Anwenden einer Hinzufüge-Operation mit einer existierenden Komponente	Nichtausführung der Operation und Ausgabe einer Meldung
Anwenden einer Hinzufüge-Op. mit nicht vorhandenem Komponenten-Typ	Nichtausführung der Operation und Ausgabe einer Meldung
Anlegen einer Verbindung	Ausführen der Operation
Anlegen einer Verbindung zu einem Port der nicht existiert	Nichtausführung der Operation und Ausgabe einer Meldung
Anlegen einer Verbindung zu zu der kein Typ existiert	Nichtausführung der Operation und Ausgabe einer Meldung
Anlegen einer Verbindung in der falschen Richtung	Nichtausführung der Operation und Ausgabe einer Meldung
Ändern eines Parameters	Ausführen der Operation
Ändern eines nicht existierenden Parameters	Nichtausführung der Operation und Ausgabe einer Meldung
Löschen einer Komponenten-Instanz	Ausführen der Operation
Löschen einer Komponenten-Instanz die nicht existiert	Nichtausführung der Operation und Ausgabe einer Meldung

Dieses Vorgehen führt dazu, dass es zu Folgefehlern kommen kann. Ist es beispielsweise nicht möglich, eine Komponente in der Implementierung anzulegen, ist es ebenso nicht möglich, Verbindungen zu dieser Komponente anzulegen. Die Applikation protokolliert alle diese Folgefehler und stellt als Ergebnis ein unvollständiges Komponentensystem zur Verfügung. Ob dieses Ergebnis genutzt werden kann, muss der Nutzer für jeden Einzelfall manuell entscheiden.

Die Aufstellung in Tabelle 6.1 soll einen Überblick über die implementierte Funktionalität geben und einen Eindruck vermitteln, wozu die Applikation verwendet werden kann. Ein Ziel ist es, die Applikation in die bestehenden Implementierungen nahtlos einzufügen und dem Nutzer die Anwendung in konkreten Aufgaben so einfach wie möglich zu gestalten. Daher sind die ausgegebenen Fehlermeldungen so aussagekräftig formuliert, dass der Nutzer genau erkennen kann, wo der Fehler liegt. So wird beispielsweise der Name der Delta-Operation und der Grund ihrer nicht ordnungsgemäßen Anwendung angegeben.

7 Diskussion der Ergebnisse

Das Ziel, ein Konzept zur Unterstützung der Wiederverwendung in komponentenbasierten Architekturen vorzulegen, konnte erreicht werden. Durch die Berücksichtigung der technischen Komponenten und der Meta-Metamodelle bei der Entwicklung der Metamodell kann die Integration in bestehende Architekturen erreicht werden (R1). Zusätzlich wurden Abläufe für die Verwendung der Modelle konform zum aktuell in der Praxis gängigen Vorgehen vorgestellt. Das Delta-Modell erlaubt die modellbasierte Beschreibung der Variabilität (R2) und die Integration von bestehenden Lösungen (R3), da die Komponenten- und Delta-Modelle leicht nachträglich erzeugt werden können.

Durch die Nutzung des Komponenten-Modells kann von der konkreten Implementierung abstrahiert (R4) werden. Das Komponenten-Modell kann zur Abbildung von hybriden Systemen verwendet werden (R13). Dadurch wird die Variabilität im Komponentensystem unabhängig von den konkreten Komponenten beschrieben. So kann das in den Komponentensystemen enthaltene Wissen genutzt werden. Die Referenz vom Typ-Modell einer Komponente zu beliebig vielen Komponenten-Typen oder anderen Vorlagen (z. B. Prototypen) ermöglicht die Abbildung von unterschiedlichen Versionen einer Komponente (R6). So sind die kompatiblen Komponenten-Typen zu einem Typ-Modell explizit beschrieben.

Das Delta-Modell beschreibt nicht nur die Transformation der Komponentensysteme, sondern auch die Beziehungen zwischen den Delta-Modellen und somit den Varianten (R5). Durch ein Feature-Modell werden Varianten von Produkten zusammengefasst. So wird der Kontext von Varianten und Produkte für (potentielle) Nutzer explizit und einfach interpretierbar dargestellt. Durch erkundbar abgelegte Modellinstanzen und das vorgestellte Metamodell wird die automatisierte Interpretation der Modelle unterstützt (R12).

Der Prozess zur Integration von geänderten Varianten bzw. neuen Versionen von Komponenten wird durch die Manager-Bausteine realisiert (R7). Die vorgestellten Referenzen zwischen den Modellen untereinander und zu den Komponentensystemen ermöglicht die genaue Verfolgung der vorliegenden Komponenten sowie ihrer Abhängigkeiten. So können die Manager-Bausteinen neue Versionen von Komponenten oder neue Varianten propagieren. Das an die Versionsverwaltung angelehnte Konzept setzt auf in Versionsverwaltungssystemen bewährte Strategien, um die geforderte Funktionalität bereitzustellen. Durch die Verwendung eines Servers und beliebig vieler Clients wird eine Integration in verteilte Systeme ermöglicht (R8). Der zentrale Server zur Bereitstellung von Varianten ist ebenso an die Versionsverwaltung angelehnt (R9). So können die zur Wiederverwendung vorgesehen Artefakte innerhalb von Organisationseinheiten verbreitet werden. Eine doppelte und somit potentiell inkonsistente Datenhaltung wird so vermieden.

Die funktionalen Anforderungen, beschrieben in Kapitel 3, werden erfüllt. Für die Wiederverwendung ist die Nutzung der Delta-Modelle in einem anderen Kontext nützlich. Das

Konzept stellt so einen Ansatz für die Verwaltung von Varianten in Komponentensystemen (z. B. Funktionsbausteinnetzen) dar.

Ebenso werden die nicht-funktionalen Anforderungen erfüllt und die Integration in bestehende Prozesse (R10) wird durch die Berücksichtigung von Prozessen aus dem Engineering wie der NA 35 oder dem ACPLT/SDP sichergestellt. Die Verwendung von Komponenten als Grundlage der Wiederverwendung ermöglicht die Nutzung bestehender Paradigmen und Sprachen (R11) aus der Automatisierungstechnik. Durch den Aufbau als Objektstrukturen und die Verwendung einer Laufzeitumgebung, die den Zugriff auf Objekte erlaubt, können die Modelle Grundlage für eine automatisierte Weiterverarbeitung sein (R12).

Delta-Modelle werden selbst bei einfachen Aufgaben sehr schnell so komplex, dass sie ohne eine automatisierte Bearbeitung schwer anzuwenden sind. Durch die Nutzung der objektorientierten Modellierung für jede Operation werden die Modelle selbst relativ groß. Andere Ansätze für die Abbildung der Delta-Modelle sind, eine andere Form der Modellierung der Deltas oder eine optimierte Form der Implementierung des vorgestellten Modells zu verwenden. Gleichwohl ändert sich für den Nutzer dadurch nicht die Menge der Informationen, die ihm bei der Anwendung des Konzepts zur Verfügung gestellt werden. Dementsprechend muss diese Informationsmenge durch ein Engineering-Tool für den Nutzer handhabbar gemacht werden. Durch eine automatisierte Informationsverarbeitung kann der Nutzer unterstützt werden.

Positiv an Delta-Modellen ist, dass die Objekte der Wiederverwendung von der Art und der Größe frei gewählt werden können. So können je nach Anwendungsfall die wiederverwendbaren Objekte möglichst gut definiert werden (vgl. [Bör89]). Das Zusammenfassen und Trennen von Delta-Modellen ermöglicht einen Aufgaben- und Zielgruppen-orientierten Einsatz von Objekten der Wiederverwendung. So entsteht eine Grundlage zu einem Wiederverwendungsmanagement in komponentenorientierten Systemen.

Im vorgestellten Konzept wurde in zwei Punkten von den Delta-Modellen in der Softwaretechnik abgewichen. Es wurde eine Reihenfolge in die Delta-Operationen integriert, da es bei der Anwendung auf hierarchische Systeme erforderlich ist, zunächst die überlagerten Komponenten anzulegen. Zusätzlich wurden die Regeln für die Anwendung der Delta-Operationen enger gefasst. So ist ist es nicht regelkonform, eine Hinzufügen-Operation auf ein Objekt anzuwenden, das bereits existiert. Dies stellt einen Fehler dar, der möglicherweise ein Indiz für ein größeres Problem ist. Daher muss der Sachverhalt protokolliert und dem Nutzer zur Kenntnis gebracht werden.

Die beschriebene Variabilität beschränkt sich auf Variabilität in den Modellen. So kann durch die Einführung einer neuen Version mit einem größeren Funktionsumfang ein neues Komponentensystem realisiert werden. Dieses wird nur dann im Modell als neue Variante dargestellt, wenn eine neue Komponente im Typ-Modell angelegt wird. Erfolgt dies nicht, entsteht zwar ein neues Produkt, allerdings wird dieses nicht modelliert. Dies kann zu einem unvollständigen Modell führen, wenn das Typ-Modell nicht aktualisiert wird.

Die Einführung des vorgestellten Konzepts erfordert neben der technischen auch eine organisatorische Umsetzung. Das bedeutet, dass die Abläufe in einer Organisationseinheit so gestaltet werden müssen, dass alle Komponentensysteme auf einem zentralen Server abgelegt werden. Den dafür nötigen Prozess zu definieren, ist relativ einfach möglich. Damit der Prozess in der täglichen Praxis verwendet und der Nutzen durch die Wiederverwendung

realisiert wird, müssen die Anwender für diesen Prozess gewonnen werden. Dies kann beispielsweise durch eine nahtlose Integration in bestehende Tools und die Herausarbeitung des Nutzens für den Einzelnen passieren.

Wird das im Rahmen dieser Arbeit entwickelte Konzept an den vorgestellten Grundsätzen zur ordnungsgemäßen Modellierung gemessen, so werden die relevanten Kriterien erfüllt. Die entwickelten Metamodelle sind semantisch und syntaktisch richtig aufgestellt. Grundlage dafür ist das SIC-Modell und das ACPLT/RTE bzw. OPC UA Metamodell. Die Modelle bilden nur die für den beschriebenen Anwendungsfall nötigen Aspekte klar und übersichtlich ab. Daher werden die Forderungen nach Klarheit und Relevanz ebenso erfüllt. Da alle benötigten Sichten auf den Anwendungsfall durch die Metamodelle adressiert werden, ist ein systematischer Aufbau gegeben. Die Forderungen nach der Wirtschaftlichkeit und der Vergleichbarkeit sind im Rahmen dieser Arbeit von untergeordneter Bedeutung. Bei der Verwendung der Modelle im Rahmen des vorgestellten Konzepts besteht allerdings durch die Wiederverwendung ein enormes Einsparpotential.

8 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein Konzept zur Unterstützung der Wiederverwendung von komponentenbasierten Architekturen vorgestellt. Für die Wiederverwendung wurden Delta-Modelle zur Beschreibung der Variabilität genutzt. Kern des Konzepts sind ein Metamodell für Komponenten und ein Metamodell für Delta-Modelle. Ausgehend von diesen Modellen wurden Transformationen und Prozesse zu deren Nutzung entwickelt. Die Delta-Modelle sollen in der dezentralen Entwicklung zur Nutzung von existierenden Lösungen verwendet werden. Dafür wurden eine Architektur und Prozesse zur Anwendung vorgestellt.

Die Erprobung des Konzepts wurde anhand einer prototypischen Implementierung demonstriert und es wurden Randbedingungen und Empfehlungen für seine Verwendung formuliert. Neben den Datenmodellen und Transformationen wurde ein Abstandsmaß für Delta-Modelle entwickelt, mit dessen Hilfe entschieden werden kann, ob Produkte Varianten voneinander sind. Zusätzlich wurde ein Vorschlag gemacht, wie durch die farbliche Kodierung der Komponenten in einer Visualisierung Varianten dargestellt werden können.

Zukünftige Arbeiten können sich mit der Anwendung des Konzepts in der Fertigungstechnik beschäftigen. Dafür muss der Aufbau der dort verwendeten Systeme und die spezifischen Prozesse dieser Domäne berücksichtigt werden. Die Nutzung von Fertigungszellen mit verschieden Systemen und einem hohen Anteil gleicher Komponenten bietet sich dafür an. In diesem Bereich existieren bereits verschiedene Ansätze, die Funktionalitäten dieser Zelle und der darin verbauten Aktoren zu klassifizieren. Die Nutzung von Komponenten-Modellen kann zu einem Nutzen von Strukturmodellen und den damit verbundenen Vorteilen für die automatisierte Interpretation der Topologie in der Fertigungsautomation führen.

Die Weiterentwicklung der Architektur für die dezentrale Wiederverwendung könnte von der Server-Client-Struktur hinzu einer Peer-to-Peer-Struktur könnte Untersucht werden. Für gewisse Anwendungsfälle könnte der Verzicht auf einen zentralen Server Vorteile bieten.

Ebenso ist die Integration von virtuellen Komponenten in ein Komponentensystem ein weiterer Ansatz, der verfolgt werden kann. Dies kann im Bereich des Testens von hybriden Systemen einen Fortschritt bedeuten oder die virtuelle Inbetriebnahme erleichtern. Im Kontext von Industrie 4.0 wird die virtuelle Inbetriebnahme als Weg zur Reduktion von Kosten und Aufwänden diskutiert.

Das Konzept kann eine Grundlage für die Verteilung von Funktionalität zwischen unterschiedlichen Rechenknoten sein. Die Varianten mit ihren unterschiedlichen Anforderungen an die Performance und den individuellen Features kann zu einer Anpassung oder Verschiebung von Last genutzt werden. Dies ist ein Ansatz, die verbaute Hardware besser auszulasten bzw. Lastspitzen durch Verteilung der Last auf andere Knoten abzufedern.

Literaturverzeichnis

[AE17] Azarmipour, Mahyar ; Epple, Ulrich: Interoperabilität von OPC UA und DDS; Nichtredigierter Manuskriptdruck. In: *Automation 2017 : technology networks processes : 18. Leitkongress der Mess- und Automatisierungstechnik : Kongresshaus Baden-Baden, 27. und 28. Juni 2017 / VDI VDE, VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik* Bd. 2293. Düsseldorf : VDI Verlag GmbH, Jun 2017 (VDI-Berichte), 87-88. – Datenträger: 1 USB-Stick 1.1

[Alb03] Albrecht, Harald: On Meta-Modeling for Communication in Operational Process Control Engineering. In: *at–Automatisierungstechnik/Methoden und Anwendungen der Steuerungs-, Regelungs-und Informationstechnik* 51 (2003), Nr. 7/2003, S. 339–340 4.4.1

[App90] Appel, Andrew W.: A runtime system. In: *Lisp and Symbolic Computation* 3 (1990), Nr. 4, S. 343–380 2.1.1

[ASE08] Abel, Dirk (Hrsg.) ; Spohr, Gerd-Ulrich (Hrsg.) ; Epple, Ulrich (Hrsg.): *Integration von Advanced Control in der Prozessindustrie: Rapid Control Prototyping*. 1. Aufl. Weinheim : Wiley-VCH, 2008. `http://dx.doi.org/10.1002/9783527626373`. `http://dx.doi.org/10.1002/9783527626373`. – ISBN 3527312056 2.1.1, 2.1.3

[Ava06] Avak, Björn: *Variant management of modular product families in the market phase* 4.3.1

[Bat05] Batory, Don: Feature models, grammars, and propositional formulas. In: *International Conference on Software Product Lines* Springer, 2005, S. 7–20 4.3.2

[BFK+17] Bloch, Henry ; Fay, Alexander ; Knohl, Torsten ; Hoernicke, Mario ; Bernshausen, Jens ; Hensel, Stephan ; Hahn, Anna ; Urbas, Leon: A microservice-based architecture approach for the automation of modular process plants. In: *Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on* IEEE, 2017, S. 1–8 2.1.1, 2.1.4, 5.1.1

[BHH+16] Bernshausen, J ; Haller, A ; Holm, T ; Hoernicke, M ; Obst, M ; Ladiges, J: Namur–Modul Type Package. Modulbeschreibung für die effiziente Automatisierung modularer Anlagen. In: *atp edition–Automatisierungstechnische Praxis* 58 (2016), S. 1–2 2.2.1

[Bör89] Börstler, Jürgen: *Wiederverwendbarkeit und Softwareentwicklung-Probleme, Lösungsansätze und Bibliographie.* RWTH, Fachgruppe Informatik, 1989 4.2, 4.2.1, 7

[Brä04] Bräutigam, Lars-Peter: *Kostenverhalten bei Variantenproduktion.* Wiesbaden : Deutscher Universitätsverlag, 2004 (Schriften zum Produktionsmanagement). http://dx.doi.org/10.1007/978-3-322-81758-7. http://dx.doi.org/10.1007/978-3-322-81758-7. – ISBN 3–8244–8109–X 4.3.1

[BRS95] Becker, Jörg ; Rosemann, Michael ; Schütte, Reinhard: Grundsätze ordnungsmäßiger modellierung. In: *Wirtschaftsinformatik* 37 (1995), Nr. 5, S. 435–445 4.4, 4.4, 4.4.1

[BSF+09] Barth, M. ; Strube, M. ; Fay, A. ; Weber, P. ; Greifeneder, J.: Object-oriented engineering data exchange as a base for automatic generation of simulation models. In: *2009 35th Annual Conference of IEEE Industrial Electronics*, 2009. – ISSN 1553–572X, S. 2465–2470 4.4.1

[BSG12] Buchholz, Meike ; Souren, Rainer ; Gelbrich, Katja: *Theorie der Variantenvielfalt: Ein produktions- und absatzwirtschaftliches Erklärungsmodell: Zugl.: Ilmenau, Techn. Univ., Diss., 2012.* Wiesbanden : Springer Gabler, 2012 (Springer Gabler Research). – ISBN 978–3–8349–4199–2 4.3.1, 4.3.1, 4.3.3

[BUN] Bundesministerium für Wirtschaft und Energie: *Plattform Industrie 4.0.* http://www.plattform-i40.de/I40/Navigation/DE/Home/home.html 1.1

[CGR+12] Czarnecki, Krzysztof ; Grünbacher, Paul ; Rabiser, Rick ; Schmid, Klaus ; Wasowski, Andrzej: Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In: *Proceedings of the 6th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'12)*, ACM, 2012, S. 173–182 4.3, 4.3.2

[CHS10] Clarke, Dave ; Helvensteijn, Michiel ; Schaefer, Ina: Abstract Delta Modeling. In: *SIGPLAN Not.* 46 (2010), Oktober, Nr. 2, 13–22. http://dx.doi.org/10.1145/1942788.1868298. – DOI 10.1145/1942788.1868298. – ISSN 0362–1340 4.3.2, 4.6, 4.3.3, 8

[CSFP08] Collins-Sussman, Ben ; Fitzpatrick, Brian W. ; Pilato, C M.: *Version control with subversion.* O'Reilly, 2008 4.2.3

[CW98] Conradi, Reidar ; Westfechtel, Bernhard: Version models for software configuration management. In: *ACM Computing Surveys (CSUR)* 30 (1998), Nr. 2, S. 232–282 4.2.3

[DB+07] Dalgarno, Mark ; Beuche, Danilo u. a.: Variant management. In: *3rd British Computer Society Configuration Management Specialist Group Conference* Bd. 1, 2007 4.3.1

[DF04] Draht, R ; Fedai, M: CAEX-ein neutrales Datenaustauschformat für Anlagendaten-Teil 1 und 2. In: *Automatisierungstechnische Praxis-atp* 46 (2004), Nr. 2, S. 52–56 2.2.1

[Die02] Dietzsch, Andreas: *Systematische Wiederverwendung in der Software-Entwicklung*. Wiesbaden and s.l. : Deutscher Universitätsverlag, 2002. http://dx.doi.org/10.1007/978-3-663-11580-9. http://dx.doi.org/10.1007/978-3-663-11580-9. – ISBN 978–3–8244–2151–0 2.1.2, 2.2.1, 2.2.1, 4.2.1

[DIN02] DIN Deutsches Institut für Normung e.V.: *Technische Produktdokumentation - CAD-Modelle, Zeichnungen und Stücklisten - Teil 1: Begriffe*. Berlin, 2002 4.3.1, 4.3.1

[DIN14] DIN Deutsches Institut für Normung e.V.: *DIN SPEC 40912: Kernmodelle - Beschreibung und Beispiele*. Berlin, 2014 2.2.1, 1, 2.2.1, 2.8, 2.2.1, 4.4, 4.7, 4.4.1, 4.4.1, 4.4.3, 4.12, 4.4.3, 4.13, 5.1.1, 8

[DLPH08] Drath, Rainer ; Luder, Arndt ; Peschke, Jorn ; Hundt, Lorenz: AutomationML-the glue for seamless automation engineering. In: *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on* IEEE, 2008, S. 616–623 4.4.2

[DMG+17] Drath, Rainer ; Malakuti, Somayeh ; Grüner, Sten ; Grothoff, Julian A. ; Wagner, Constantin A. ; Epple, Ulrich ; Hoffmeister, Michael ; Zimmermann, Patrick: Die Rolle der Industrie 4.0 „Verwaltungsschale“ und des „digitalen Zwillings“ im Lebenszyklus einer Anlage : Navigationshilfe, Begriffsbestimmung und Abgrenzung. In: *Automation 2017 : technology networks processes : 18. Leitkongress der Mess- und Automatisierungstechnik : Kongresshaus Baden-Baden, 27. und 28. Juni 2017 / VDI VDE, VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik* Bd. 2293. Düsseldorf : VDI Verlag GmbH, Jun 2017 (VDI-Berichte), 93-94. – Datenträger: 1 USB-Stick 1.1, 2.1.3, 2.2.1, 4.2.4

[Dud] Dudenredaktion ; Dudenverlag (Hrsg.): *Version*. https://www.duden.de/node/703510/revisions/1668452/view 4.2.3

[EE13] Evertz, Lars ; Epple, Ulrich: Laying a basis for service systems in process control. In: *18th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2013, S. 1–8 1.1, 2.1.4, 4.4.3

[EMPA17] Epple, Ulrich ; Mertens, Martin ; Palm, Florian ; Azarmipour, Mahyar: Using Properties as Semantic Base for Interoperability. In: *IEEE transactions on industrial informatics* (2017), 9 Seiten. http://dx.doi.org/10.1109/TII.2017.2741339. – DOI 10.1109/TII.2017.2741339. – ISSN 1941–0050. – Online-First 3.2, 4.4.3

[Ens01] Enste, Udo: *Fortschritt-Berichte VDI Reihe 8, Meß-, Steuerungs- und Regelungstechnik*. Bd. 884: *Generische Entwurfsmuster in der Funktionsbausteintechnik und deren Anwendung in der operativen Prozeßführung: Zugl.: Aa-*

chen, Techn. Hochsch., Diss., 2000. Als Ms. gedr. Düsseldorf : VDI-Verl., 2001. – ISBN 3183884089 2.2.1, 4.1, 5.1.3

[Epp08] Epple, Ulrich: Begriffliche Grundlagen der leittechnischen Modellwelt. In: *Automatisierungstechnische Praxis : atp* 50 (2008), Nr. 4, 83-91. http://publications.rwth-aachen.de/record/133133. – ISSN 0178–2320 2.2.1, 4.4

[FA09] Früh, Karl F. (Hrsg.) ; Ahrens, Wolfgang (Hrsg.): *Handbuch der Prozessautomatisierung: Prozessleittechnik für verfahrenstechnische Anlagen*. 4., überarb. Aufl. München : Oldenbourg, 2009. – ISBN 9783835631427 2.1, 2.1.1, 2.1.1, 3.1

[FFVH12] Feldmann, Stefan ; Fuchs, Julia ; Vogel-Heuser, Birgit: Modularity, variant and version management in plant automation–future challenges and state of the art. In: *DS 70: Proceedings of DESIGN 2012, the 12th International Design Conference, Dubrovnik, Croatia*, 2012 4.2.3, 4.2.4

[FLK+14] Fuchs, Julia ; Legat, Christoph ; Kernschmidt, Konstantin ; Frank, Timo ; Vogel-Heuser, Birgit: Interdisziplinärer Produktlinienansatz zur Unterstützung der Wiederverwendbarkeit im Maschinen-und Anlagenbau. In: *13. Fachtagung: Entwurf komplexer Automatisierungssysteme (EKA 2014)*, 2014 4.2.4

[FR07] France, Robert ; Rumpe, Bernhard: Model-driven development of complex software: A research roadmap. In: *2007 Future of Software Engineering* IEEE Computer Society, 2007, S. 37–54 4.4.2

[FVHF+15] Fay, Alexander ; Vogel-Heuser, Birgit ; Frank, Timo ; Eckert, Karin ; Hadlich, Thomas ; Diedrich, Christian: Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns. In: *Journal of Systems and Software* 101 (2015), S. 221–235. http://dx.doi.org/10.1016/j.jss.2014.12.028. – DOI 10.1016/j.jss.2014.12.028 1.1

[Gas12] Gasser, Tom M. (Hrsg.): *Berichte der Bundesanstalt für Strassenwesen F, Fahrzeugtechnik*. Bd. 83: *Rechtsfolgen zunehmender Fahrzeugautomatisierung: Gemeinsamer Schlussbericht der Projektgruppe ; Bericht zum Forschungsprojekt F 1100.5409013.01*. Bremerhaven : Wirtschaftsverl. NW Verl. für neue Wiss, 2012 http://bast.opus.hbz-nrw.de/volltexte/2012/587/. – ISBN 978–3–86918–189–9 2.1

[GE13] Grüner, Sten ; Epple, Ulrich: Paradigms for unified runtime systems in industrial automation. In: *Control Conference (ECC), 2013 European*, 2013, 3925–3930 6.1

[GHJV11] Gamma, Erich ; Helm, Richard ; Johnson, Ralph E. ; Vlissides, John: *Design patterns: Elements of reusable object-oriented software*. 39. printing. Boston : Addison-Wesley, 2011 (Addison-Wesley professional computing series). – ISBN 0201633612 4.4.1

[GJHV11] Gamma, Erich ; Johnson, Ralph ; Helm, Richard ; Vlissides, John: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Pearson Deutschland GmbH, 2011 4.2.2

[GPE16] Grothoff, Julian A. ; Palm, Florian ; Epple, Ulrich: Modelltransformation als Softwareadapter für OPC Unified Architecture. In: *7. Jahreskolloquium "Kommunikation in der Automation": 30.11.2016 : KommA 2016 Kommunikation in der Automation / Jürgen Jasperneite, Ulrich Jumar (Hrsg.) ; eine Kooperation von: inIT, IFAK*. Lemgo, 2016 4.4.1

[GPP16] Grüner, Sten ; Pfrommer, Julius ; Palm, Florian: RESTful Industrial Communication with OPC UA. In: *IEEE Transactions on Industrial Informatics* (2016), S. 1. http://dx.doi.org/10.1109/TII.2016.2530404. – DOI 10.1109/TII.2016.2530404. – ISSN 1551–3203 2.2.2, 6.1

[GR95] Goldberg, Adele ; Rubin, Kenneth S.: Succeeding with Objects. Decision Frameworks for Project Management. In: *Reading, Mass.: Addison-Wesley,| c1995* (1995) 4.2.1

[Grü17] Grüner, Sten: *Ressourcenadaptive Anwendungen für die operative Prozessleittechnik*, RWTH Aachen, Dissertation, 2017 2.1.1, 2.1.1, 2.1.1

[Har87] Harel, David: Statecharts: A visual formalism for complex systems. In: *Science of computer programming* 8 (1987), Nr. 3, S. 231–274 5.4.2

[HC01] Heineman, George T. ; Councill, William T.: *Component-based software engineering: Putting the pieces together*. Boston, Mass. : Addison-Wesley, 2001. – ISBN 0–201–70485–4 2.2.1, 2.10, 8

[HHK+13] Haber, Arne ; Hölldobler, Katrin ; Kolassa, Carsten ; Look, Markus ; Rumpe, Bernhard ; Müller, Klaus ; Schaefer, Ina: Engineering delta modeling languages. In: *Proceedings of the 17th International Software Product Line Conference* ACM, 2013, S. 22–31 4.3.3

[HKR+11] Haber, Arne ; Kutz, Thomas ; Rendel, Holger ; Rumpe, Bernhard ; Schaefer, Ina: Delta-oriented architectural variability using monticore. In: *Proceedings of the 5th European Conference on Software Architecture: Companion Volume* ACM, 2011, S. 6 4.3.3

[HMW12] Helvensteijn, Michiel ; Muschevici, Radu ; Wong, Peter Y.: Delta modeling in practice: a Fredhopper case study. In: *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems* ACM, 2012, S. 139–148 4.3.3

[HRRS11] Haber, Arne ; Rendel, Holger ; Rumpe, Bernhard ; Schaefer, Ina: Delta Modeling for Software Architectures. In: *Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme VII* Citeseer, 2011 4.3.3

[HSF+13] Holm, Thomas ; Schröck, Sebastian ; Fay, Alexander ; Jäger, Tobias ; Löwen, Ulrich: Engineering von "Mechatronik und Software" in automatisierten Anlagen: Anforderungen und Stand der Technik. In: *Software Engineering (Workshops)*, 2013, S. 261–272 2.1, 2.1.3, 2.1.3, 2.2.2, 3.1

[IEC04] International Electrotechnical Commission: *IEC 61131:Programmable controllers - Part 1: General information* . 2004 2.3, 3.1, 5.1.2, 8

[IEC05] International Electrotechnical Commission: *IEC 61499: Function blocks*. 2005 2.1.1, 2.1.1, 2.1.4, 2.2.1, 2.2.2

[IEC10] International Electrotechnical Commission: *IEC 62541. OPC Unified Architecture Part 1-10, Release 1.0*. 2010 2.2.2, 4.4, 4.4.1, 4.4.1

[IEC14a] International Electrotechnical Commission: *IEC 61131: Internationales Elektrotechnisches Wörterbuch - Teil 351: Leittechnik*. 2014 4.4

[IEC14b] International Electrotechnical Commission: *IEC 61131:Programmable controllers - Part 3: Programming languages*. 3rd. 2014 5, 2.1.1, 2.4, 2.2.2, 5.1.4, 5.3.3, 8

[IEC14c] International Electrotechnical Commission: *IEC 62714: Engineering data exchange format for use in industrial automation systems engineering - Automation markup language - Part 1: Architecture and general requirements*. Juni 2014 4.4.2

[IEC16] International Electrotechnical Commission: *IEC 62424: Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools*. Juli 2016 2.2.1, 2.2.1, 2.2.2, 3.2, 4.4.2, 5.2.4, 5.3.2

[IEC17] International Electrotechnical Commission: *IEC 61360:Standard data element types with associated classification scheme - Part 1: Definitions - Principles and methods*. 3rd. 2017 4.4.3

[ISO11] International Electrotechnical Commission: *ISO/IEC 25010: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*. 2011 4.1, 4.2

[ISO12] International Electrotechnical Commission: *ISO/IEC 19505-1: Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure*. April 2012 4.4.2

[JT00] John, Karl-Heinz ; Tiegelkamp, Michael: *SPS-Programmierung mit IEC 61131-3: Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen*. 3., neubearb. Aufl. Berlin and Heidelberg and New York and Barcelona and Hongkong and London and Mailand and Paris and Singapur and Tokio : Springer, 2000 (VDI-Buch). – ISBN 3540664459 2.1.1, 2.1.1, 2.2.2

[Kam17] Kampert, David: *Operative Verwendung merkmalbasierter Information in der Automatisierung; Als Manuskript gedruckt.* Düsseldorf, RWTH Aachen University, Dr., 2017. http://dx.doi.org/10.18154/RWTH-2017-06589. – DOI 10.18154/RWTH–2017–06589. – 1 Online–Ressource (X, 124 Seiten) : Illustrationen, Diagramme S. – Auch veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Dissertation, RWTH Aachen University, 2017 2.1.1, 4.4

[KBD+08] Krämer, Stefan ; Bamberg, Andreas ; Dünnebier, Guido ; Hagenmeyer, Veit ; Piechottka, Uwe ; Schmitz, S: Prozessführung: Beispiele, Erfahrung und Entwicklung. In: *Chemie Ingenieur Technik* 80 (2008), Nr. 9, S. 1341–1342 2.1

[KCH+90] Kang, Kyo C. ; Cohen, Sholom G. ; Hess, James A. ; Novak, William E. ; Peterson, A S.: Feature-oriented domain analysis (FODA) feasibility study / Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst. 1990. – Forschungsbericht 4.3.2

[KCH+92] Kang, Kyo C. ; Cohen, Sholom ; Holibaugh, Robert ; Perry, James ; Peterson, A S.: A reuse-based software development methodology / CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST. 1992. – Forschungsbericht 4.2.1

[KCJ+10] Karnouskos, Stamatis ; Colombo, Armando W. ; Jammes, Francois ; Delsing, Jerker ; Bangemann, Thomas: Towards an architecture for service-oriented process monitoring and control. In: *IECON 2010 - 36th Annual Conference of IEEE Industrial Electronics*, 2010, S. 1385–1391 2.2, 2.1.1, 8

[KE12] Kampert, David ; Epple, Ulrich: Kernmodelle für die Systembeschreibung - Ein Konzept zur Vereinfachung. In: *Atp-Edition : automatisierungstechnische Praxis* 54 (2012), Nr. 7/8, 40-48. http://publications.rwth-aachen.de/record/140375. – ISSN 0178–2320 2.1

[KLD02] Kang, Kyo C. ; Lee, Jaejoon ; Donohoe, Patrick: Feature-Oriented Product Line Engineering. In: *IEEE software* 19 (2002), Nr. 4, S. 58–65 4.3.2

[KLL+14] Kowal, Matthias ; Legat, Christoph ; Lorefice, David ; Prehofer, Christian ; Schaefer, Ina ; Vogel-Heuser, Birgit: Delta modeling for variant-rich and evolving manufacturing systems. In: Nair, Anil R. (Hrsg.) ; Prähofer, Herbert (Hrsg.) ; Zoitl, Alois (Hrsg.) ; Jetley, Raoul (Hrsg.) ; Dubey, Alpana (Hrsg.) ; Kumar, Atul (Hrsg.): *the 1st International Workshop on Modern Software Engineering Methods for Industrial Automation*, 2014, S. 32–41 4.3.1, 4.3.3

[Koe85] Koen, Billy V.: *Definition of the Engineering Method.* American Sisiety for Engineering Education, 1985 2.1.3

[KPST14] Kowal, Matthias ; Prehofer, Christian ; Schaefer, Ina ; Tribastone, Mirco: Model-based Development and Performance Analysis for Evolving

Manufacturing Systems. In: *at - Automatisierungstechnik* 62 (2014), Nr. 11. http://dx.doi.org/10.1515/auto-2014-1098. – DOI 10.1515/auto–2014–1098. – ISSN 0178–2312 4.3.3

[Kru95] Kruchten, Philippe B.: The 4+ 1 view model of architecture. In: *IEEE software* 12 (1995), Nr. 6, S. 42–50 4.11, 4.4.2, 8

[Kru04] Kruchten, Philippe: *The rational unified process: an introduction*. Addison-Wesley Professional, 2004 2.2.1, 4.4, 4.4.2

[Lee08] Lee, Edward A.: Cyber physical systems: Design challenges. In: *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, 2008, S. 363–369 1.1

[Lim94] Lim, W. C.: Effects of reuse on quality, productivity, and economics. In: *IEEE Software* 11 (1994), Nr. 5, S. 23–30. http://dx.doi.org/10.1109/52.311048. – DOI 10.1109/52.311048. – ISSN 0740–7459 2.1.2, 4.2, 4.2, 4.3.3

[Lin94] Lingnau, Volker: *Betriebswirtschaftliche Studien*. Bd. 58: *Variantenmanagement: Produktionsplanung im Rahmen einer Produktdifferenzierungsstrategie: Zugl.: Berlin, Techn. Univ., Diss., 1994*. Berlin : Schmidt, 1994. – ISBN 3–503–03619–9 4.3.1, 4.3.1

[LKS16] Lity, Sascha ; Kowal, Matthias ; Schaefer, Ina: Higher-order Delta Modeling for Software Product Line Evolution. In: *Proceedings of the 7th International Workshop on Feature-Oriented Software Development*. New York, NY, USA : ACM, 2016 (FOSD 2016). – ISBN 978–1–4503–4647–4, 39–48 4.3.3

[LM06] Leitner, Stefan-Helmut ; Mahnke, Wolfgang: OPC UA–service-oriented architecture for industrial applications. (2006). http://www2.cs.uni-paderborn.de/cs/ag-engels/GI/ORA2006-Papers/leitner-final.pdf 4.9, 8

[Löf11] Löffler, Carina: *IPA-IAO Forschung und Praxis*. Bd. 519: *Systematik der strategischen Strukturplanung für eine wandlungsfähige und vernetzte Produktion der variantenreichen Serienfertigung: Zugl.: Stuttgart, Univ., Diss., 2011*. Heimsheim : Jost-Jetter, 2011 http://nbn-resolving.de/urn:nbn:de:bsz:93-opus-70492. – ISBN 978–3–939890–90–4 2.1.4

[LS17] Lackes, Richard ; Siepermann, Markus ; Springer Gabler Verlag (Hrsg.): *Gabler Wirtschaftslexikon*. http://wirtschaftslexikon.gabler.de/Archiv/74918/wiederverwendbarkeit-v8.html. Version: 2017 4.3

[LU17] Leon Urbas, Christian K. u.: Namur Open Architecture. In: *atp edition* 59 (2017), Nr. 01-02, 20–37. http://dx.doi.org/10.17560/atp.v59i01-02.620. – DOI 10.17560/atp.v59i01–02.620. – ISSN 2364–3137 1.1

[Lun03] Lunze, Jan: *Automatisierungstechnik: Methoden für die Überwachung und Steuerung kontinuierlicher und ereignisdiskreter Systeme ; ... mit 74 Anwendungsbeispielen und 84 Übungsaufgaben*. München : Oldenbourg, 2003. – ISBN 3486274309 2.1, 3.1

[Mah14] Mahler, Carsten: Automatisierungsmodule für ein funktionsorientiertes Automatisierungsengineering. (2014) 4.2.4, 4.5

[MB00] Mcfarlane, Duncan C. ; Bussmann, Stefan: Developments in holonic production planning and control. In: *Production Planning & Control* 11 (2000), Nr. 6, S. 522–536. http://dx.doi.org/10.1080/095372800414089. – DOI 10.1080/095372800414089. – ISSN 0953–7287 1.1

[Mer12] Mertens, Martin: *Fortschritt-Berichte VDI / Reihe 8 Mess-, Steuerungs- und Regelungstechnik*. Bd. 1207: *Verwaltung und Verarbeitung merkmalbasierter Informationen: Vom Metamodell zur technologischen Realisierung: Aachen, Techn. Hochsch., Diss., 2011*. Aachen : Hochschulbibliothek Rheinisch-Westfälische Technischen Hochschule Aachen, 2012 http://nbn-resolving.de/urn:nbn:de:hbz:82-opus-39896. – ISBN 9783185207082 4.4, 4.4.3

[Mey88] Meyer, Bertrand: *Object-oriented software construction*. Bd. 2. Prentice hall New York, 1988 1.1, 4.2

[Mey09] Meyer, Bertrand: *Object-oriented software construction*. 2. ed., 15. print. Upper Saddle River, NJ : Prentice Hall PTR, 2009. – ISBN 0136291554 1.2, 4.2, 4.2

[MPH+07] Metzger, Andreas ; Pohl, Klaus ; Heymans, Patrick ; Schobbens, Pierre-Yves ; Saval, Germain: Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis. In: *15th IEEE International Requirements Engineering Conference (RE 2007)*, 2007, S. 243–253 4.3.2

[NAM96] NAMUR Arbeitskreis 1.6: *NA 63 - Package Units*. 1996 2.1.2, 2.2.2

[NAM02] NAMUR Arbeitskreis 1.9: *NE 58 - Abwicklung von qualifizierungspflichtigen PLT - Projekten*. 2002 2.1.1

[NAM03] NAMUR Arbeitskreis 1.1: *NA 35 - Abwicklung von PLT-Projekten*. 2003 2.1.3, 2.6, 2.1.3, 8

[NAM08] NAMUR Arbeitskreis 1.11: *NE 121 - Qualitätssicherung leittechnischer Systeme*. 2008 2.1.3, 3.1

[NAM13] NAMUR Arbeitskreis 1.12: *NE 148 - Anforderungen an die Automatisierungstechnik durch die Modularisierung verfahrenstechnischer Anlagen*. 2013 2.1.2

[NAM14] NAMUR Arbeitskreis 2.2: *NE 152 - Regelgütemanagement: Überwachung und Optimierung der Basisregelung von Produktionsanlagen*. 2014 3.1

[NAM16] NAMUR Arbeitskreis 1.11: *NE 160 - Ein Referenzmodell für allgemeine Prozedurbeschreibungen*. 2016 4.4, 4.4.3

[ODU13] Obst, Michael ; Doherr, Falk ; Urbas, Leon: Wissensbasiertes Assistenzsystem für modulares Engineering. In: *at - Automatisierungstechnik* 61 (2013), Nr. 2, S. 103–108. http://dx.doi.org/10.1524/auto.2013.0011. – DOI 10.1524/auto.2013.0011. – ISSN 0178–2312 4.2.4, 4.5

[OMG15] Object Managment Group: *OMG Unified Modeling Language*. http://www.omg.org/spec/UML/2.5. Version: 2015. – Version 2.5 2.2.1, 2.2.1, 2.9, 8

[OMG16] Object Managment Group: *Meta Object Facility (MOF) 2.5.1 Core Specification*. http://www.omg.org/spec/MOF/2.5.1. Version: 2016. – Version 2.5.1 4.4.1, 4.8, 4.4.1, 8

[ORA] ORACLE: *SVN Workflow*. https://docs.oracle.com/middleware/1212/core/MAVEN/config_svn.htmMAVEN8826 4.2.3, 4.1, 8

[Ott09] Otte, Stefan: Version control systems. In: *Computer Systems and Telematics* (2009) 4.2.3

[PBL05] Pohl, Klaus ; Böckle, Günter ; Linden, Frank: *Software Product Line Engineering: Foundations, Principles, and Techniques*. Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2005. http://dx.doi.org/10.1007/3-540-28901-1. http://dx.doi.org/10.1007/3-540-28901-1. – ISBN 3–540–24372–0 4.3.1, 4.2, 4.3.1, 4.4, 4.3.2, 8

[PE94] Polke, Martin ; Epple, Ulrich: *Prozessleittechnik: Mit 8 Tabellen*. 2., völlig überarb. und stark erw. Aufl. München und Wien : Oldenbourg, 1994. – ISBN 3486225499 2.1, 2.1.1, 2.2.2

[PE17] Palm, Florian ; Epple, Ulrich: openAAS - Die offene Entwicklung der Verwaltungsschale. In: *Automation 2017 : technology networks processes : 18. Leitkongress der Mess- und Automatisierungstechnik : Kongresshaus Baden-Baden, 27. und 28. Juni 2017 / VDI VDE Gesellschaft Mess- und Automatisierungstechnik* Bd. 2293. Düsseldorf : VDI Verlag GmbH, Jun 2017 (Verein Deutscher Ingenieure: VDI-Berichte), 103-104. – Datenträger: 1 USB-Stick 2.1.4, 2.1.4, 2.2.2, 4.4.3

[PGGS16] Pfrommer, Julius ; Grüner, Sten ; Goldschmidt, Thomas ; Schulz, Dirk: A common core for information modeling in the Industrial Internet of Things. In: *at - Automatisierungstechnik* 64 (2016), Nr. 9. http://dx.doi.org/10.1515/auto-2016-0071. – DOI 10.1515/auto–2016–0071. – ISSN 0178–2312 4.4

[PGP+15] Palm, Florian ; Grüner, Sten ; Pfrommer, Julius ; Graube, Markus ; Urbas, Leon: Open Source as Enabler for OPC UA in Industrial Automation. In: *Proceedings of 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2015. – ISBN 978–1–4673–7929–8 1.1

[PSU+14] Pfrommer, Julius ; Schleipen, Miriam ; Usländer, Thomas ; Epple, Ulrich ; Heidel, Roland ; Urbas, Leon ; Sauer, Olaf ; Beyerer, Jürgen: Begrifflichkeiten um Industrie 4.0 : Ordnung im Sprachwirrwarr. In: *Entwurf komplexer Automatisierungssysteme - EKA 2014 : Beschreibungsmittel, Methoden, Werkzeuge und Anwendungen ; 13. Fachtagung mit Tutorium, 14. bis 15. Mai 2014 in Magdeburg / Ulrich Jumar; Christian Diedrich (Hrsg.).* Magdeburg : ifak Institut für Automation und Kommunikation e.V., 2014, 8 S. 4.4.1

[Rie17] Riedel, Maik: Ein Beitrag zur wissensbasierten Unterstützung bei der Auswahl technischer Ressourcen: Repräsentation und Auswertung von Prinziplösungen auf Basis multidimensionaler, heterogener, vernetzter Merkmalräume. (2017) 5.1.1

[SAG+17] Schuh, Günther ; Anderl, Reiner ; Gausemeier, Jürgen ; Ten Hompel, Michael ; Wahlster, Wolfgang: *Industrie 4.0 Maturity Index: Die digitale Transformation von Unternehmen gestalten.* Herbert Utz Verlag, 2017 1.1, 1.1, 2.1.4, 2.2.2

[Sam97] Sametinger, Johannes: *Software engineering with reusable components: With 26 tables.* Berlin : Springer, 1997. – ISBN 3–540–62695–6 2.2.1

[SBB+10] Schaefer, Ina ; Bettini, Lorenzo ; Bono, Viviana ; Damiani, Ferruccio ; Tanzarella, Nico: Delta-oriented programming of software product lines. In: *International Conference on Software Product Lines* Springer, 2010, S. 77–91 4.3.3

[Sch10] Schaefer, Ina: Variability Modelling for Model-Driven Development of Software Product Lines. In: *VaMoS* 10 (2010), S. 85–92 4.3.3, 4.3.3

[Sch16a] Schüller, Andreas: *Ein Referenzmodell zur Beschreibung allgemeiner Prozeduren im leittechnischen Umfeld.* Düsseldorf, RWTH Aachen University, Dissertation, 2016. `http://publications.rwth-aachen.de/record/686692`. – XIV, 148 Seiten : Diagramme S. – Als Manuskript gedruckt. - Weitere Reihe: Lehrstuhl für Prozessleittechnik der RWTH Aachen; Dissertation, RWTH Aachen University, 2016 4.4, 6.1

[Sch16b] Schröck, Sebastian: *Interdisziplinäre Wiederverwendung im Engineering automatisierter Anlagen: Anforderungen, Konzept und Umsetzungen für die Prozessindustrie.* VDI Verlag GmbH, 2016 2.1.3, 4.2.1, 4.2.2, 4.2.4, 4.3.1, 4.3.1, 4.3.2

[Sch18] Schaefer, Ina: A Personal History of Delta Modelling. In: Müller, Peter (Hrsg.) ; Schaefer, Ina (Hrsg.): *Principled Software Development: Essays Dedicated to Arnd Poetzsch-Heffter on the Occasion of his 60th Birthday.* Cham : Springer International Publishing, 2018. – ISBN 978–3–319–98046–1 4.5

[SCZ+16] Shi, Weisong ; Cao, Jie ; Zhang, Quan ; Li, Youhuizi ; Xu, Lanyu: Edge Computing: Vision and Challenges. In: *IEEE Internet of Things Journal* 3 (2016), S. 637–646 2.1.4

[SDM95] Steyaert, Patrick ; De Meuter, Wolfgang: A marriage of class-and object-based inheritance without unwanted children. In: *European Conference on Object-Oriented Programming* Springer, 1995, S. 127–144 4.2.2

[SEE09] Schlütter, M. ; Epple, U. ; Edelmann, T.: On Service-Orientation as a New Approach for Automation Environments. In: *Proceedings MATHMOD 09 Vienna* (2009) 1.1, 8

[SLU88] Stein, Lynn A. ; Lieberman, Henry ; Ungar, David: *A shared view of sharing: the treaty of Orlando*. Brown University, Department of Computer Science, 1988 4.2.2

[SRC+12] Schaefer, Ina ; Rabiser, Rick ; Clarke, Dave ; Bettini, Lorenzo ; Benavides, David ; Botterweck, Goetz ; Pathak, Animesh ; Trujillo, Salvador ; Villela, Karina: Software diversity: State of the art and perspectives. In: *International Journal on Software Tools for Technology Transfer* 14 (2012), Nr. 5, S. 477–495. `http://dx.doi.org/10.1007/s10009-012-0253-y`. – DOI 10.1007/s10009–012–0253–y. – ISSN 1433–2779 4.3.1, 4.3.2, 4.3.2, 4.3.2, 4.3.2, 4.3.2, 4.3.3

[SRVK10] Sprinkle, Jonathan ; Rumpe, Bernhard ; Vangheluwe, Hans ; Karsai, Gabor: 3 Metamodelling. In: *Model-Based Engineering of Embedded Real-Time Systems*. Springer, 2010, S. 57–76 4.4.1, 4.4.1

[SSS17] Schuster, Sven ; Seidl, Christoph ; Schaefer, Ina: Towards a Development Process for Maturing Delta-oriented Software Product Lines. In: *Proceedings of the 8th ACM SIGPLAN International Workshop on Feature-Oriented Software Development*. New York, NY, USA : ACM, 2017 (FOSD 2017). – ISBN 978–1–4503–5518–6, 41–50 4.3.3

[TE18] Trotha, Christian ; Epple, Ulrich: Assistenzsysteme in der Prozessindustrie : Ein Versuch der Klassifikation. In: *[19. Leitkongress der Mess- und Automatisierungstechnik, 2018-07-03 - 2018-07-04, Baden-Baden, Germany]* 19. Leitkongress der Mess- und Automatisierungstechnik, Baden-Baden (Germany), 3 Jul 2018 - 4 Jul 2018, 2018, 529-542 2.1

[UDKO12] Urbas, Leon ; Doherr, Falk ; Krause, Annett ; Obst, Michael: Modularisierung und Prozessführung. In: *Chemie Ingenieur Technik* 84 (2012), Nr. 5, S. 615–623. `http://dx.doi.org/10.1002/cite.201200034`. – DOI 10.1002/cite.201200034. – ISSN 0009286X 2.1.1, 4.2.4

[VDI95] Verein Deutscher Ingenieure: *VDI/VDE 3696 - Herstellerneutrale Konfigurierung von Prozeßleitsystemen* . 1995. – zrückgezogen 6.2.1

[VG07] Voelter, Markus ; Groher, Iris: Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In: *11th International Software Product Line Conference (SPLC 2007)*, 2007, S. 233–242 4.3.1, 4.3.2

[VH09] Vogel-Heuser, B: Visions of automation engineering in 2020. In: *Automation Technology in Practice (atp)* (2009) 3.2, 4.2.3

[VHDB13] Vogel-Heuser, Birgit ; Diedrich, Christian ; Broy, Manfred: Anforderungen an CPS aus Sicht der Automatisierungstechnik. In: *at – Automatisierungstechnik* 61 (2013), Nr. 10. `http://dx.doi.org/10.1515/auto.2013.0061`. – DOI 10.1515/auto.2013.0061. – ISSN 0178–2312 2.1, 2.1, 2.1.3, 2.1.4, 3.1, 8

[VHDF+14] Vogel-Heuser, Birgit ; Diedrich, Christian ; Fay, Alexander ; Jeschke, Sabine ; Kowalewski, Stefan ; Wollschlaeger, Martin ; Göhner, Peter: Challenges for Software Engineering in Automation. In: *Journal of Software Engineering and Applications* 07 (2014), Nr. 05, S. 440–451. `http://dx.doi.org/10.4236/jsea.2014.75041`. – DOI 10.4236/jsea.2014.75041. – ISSN 1945–3116 1.1, 2.1, 2.1.3, 2.1.4, 2.2.2, 2.2.2, 3.1, 4.4.1

[VHDFG13] Vogel-Heuser, Birgit ; Diedrich, Christian ; Fay, Alexander ; Göhner, Peter: Anforderungen an das Software-Engineering in der Automatisierungstechnik. In: *Software Engineering*, 2013, S. 51–66 2.1, 8

[VHON18] Vogel-Heuser, B. ; Ocker, F. ; Neumann, E. M.: Maturity variations of PLC-based control software within a company and among companies from the same industrial sector. In: *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, 2018, S. 283–290 3.2, 4.5

[VWB+09] Vajna, Sandor ; Weber, Christian ; Bley, Helmut ; Zeman, Klaus ; Hehenberger, Peter: Grundlagen der Modellbildung. In: *CAx für Ingenieure* (2009), S. 97–157 4.4

[WE15a] Wagner, Constantin ; Epple, Ulrich: Sprechende Kommandos als Grundlage moderner Prozessführungsschnittstellen. In: *AUTOMATION 2015*. Baden-Baden, 2015 2.1.1, 2.1.4, 4.1, 6.1, 6.5, 8

[WE15b] Wagner, Constantin ; Epple, Ulrich: Variant Management for Control Blocks. In: *Proceedings of the IEEE 20th International Conference on Emerging Technologies and Factory Automation*. Piscataway, NJ : IEEE, Sep 2015 4.1, 4.5, 6.2.1, 8

[WE17] Wagner, Constantin ; Epple, Ulrich: Integration von Serviceschnittstellen in Funktionsbausteinarchitekturen; Nichtredigierter Manuskriptdruck. In: *Automation 2017 : technology networks processes : 18. Leitkongress der Mess- und Automatisierungstechnik : Kongresshaus Baden-Baden, 27. und 28. Juni 2017 / VDI VDE, VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik* Bd. 2293. Düsseldorf : VDI Verlag GmbH, Jun 2017 (VDI-Berichte), 37-38. – Datenträger: 1 USB-Stick 1.1, 2.1.4, 4.1, 5.1.1, 6.1, 6.2.2, 6.6, 8

[Web14] Weber, Klaus H.: *Engineering verfahrenstechnischer Anlagen: Praxishandbuch mit checklisten und beispielen*. Heidelberg : Springer Vieweg, 2014 (VDI-Buch). http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=846203. – ISBN 978–3–662–43528–1 1.1, 2.1.3, 2.1.3, 3.1

[WES87] Woodfield, S. N. ; Embley, D. W. ; Scott, D. T.: Can Programmers Reuse Software? In: *IEEE Software* 4 (1987), July, Nr. 4, S. 52–59. http://dx.doi.org/10.1109/MS.1987.231064. – DOI 10.1109/MS.1987.231064. – ISSN 0740–7459 4.2

[WGE16] Wagner, Constantin ; Grüner, Sten ; Epple, Ulrich: Portabilität und Wiederverwendbarkeit von auf Funktionsbausteinnetzwerken basierenden Anwendungen. In: *Entwurf komplexer Automatisierungssysteme*. Magdeburg, 2016. – ISBN 978–3–944722–35–1 4.1, 4.2.2, 4.2.4

[WGE+17] Wagner, Constantin ; Grothoff, Julian ; Epple, Ulrich ; Drath, Rainer ; Malakuti, Somayeh ; Gruner, Sten ; Hoffmeister, Michael ; Zimermann, Patrick: The role of the Industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. In: *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, S. 1–8 2.1.3, 2.1.4, 2.7, 2.2.1, 2.2.1, 3.1, 4.1, 4.4.1, 4.4.2, 8

[WGE+18] Wagner, Constantin ; Grothoff, Julian ; Epple, Ulrich ; Grüner, Sten ; Wenger, Monika ; Zoitl, Alois: Ein Beitrag zu einem einheitlichen Engineering für Laufzeitumgebungen. In: *Automation 2018 : Seamless Convergence of Automation & IT : 19. Leitkongress der Mess- und Automatisierungstechnik* , 2018 4.4.3, 4.14, 5.1.1, 8

[WGKO15] Wahler, Michael ; Gamer, Thomas ; Kumar, Atul ; Oriol, Manuel: FASA: A software architecture and runtime framework for flexible distributed automation systems. In: *Journal of Systems Architecture* 61 (2015), Nr. 2, S. 82–111 4.4.1

[WKS+16] Wagner, Constantin ; Kampert, David ; Schüller, Andreas ; Palm, Florian ; Grüner, Sten ; Epple, Ulrich: Model based synthesis of automation functionality. In: *at - Automatisierungstechnik* 64 (2016), Nr. 3. http://dx.doi.org/10.1515/auto-2015-0094. – DOI 10.1515/auto–2015–0094. – ISSN 0178–2312 1.1, 3.2, 4.1, 4.4.1, 4.4.1, 4.10, 6.1, 8

[WSFE16] Wagner, Constantin ; Schüller, Andreas ; Fleischacker, Christopher ; Epple, Ulrich: An Educational Framework for Process Control Theory and Engineering Tools. In: *[The 11th IFAC Symposium on Advances in Control Education, 01.06.2016-03.06.2016, Bratislava, Slovakia]* The 11th IFAC Symposium on Advances in Control Education, Bratislava (Slovakia), 1 Jun 2016 - 3 Jun 2016, 2016 4.1

[WTE+17] Wagner, Constantin ; Trotha, Christian W. ; Epple, Ulrich ; Metzul, Alfred ; Debus, Kai ; Christoph, Helle: Requirements for the Next Generation Automation Solution of Rolling Mills. In: *[43rd Annual Conference*

of the IEEE Industrial Electronics Society (IES), IECON 2017, 2017-10-29 - 2017-11-01, Peking, Peoples R China] 43rd Annual Conference of the IEEE Industrial Electronics Society (IES), Peking (Peoples R China), 29 Oct 2017 - 1 Nov 2017, 2017 3.1, 4.1, 4.3.1

[WTPE17] Wagner, Constantin ; Trotha, Christian von ; Palm, Florian ; Epple, Ulrich: Fundamentals for the next Generation of Automation Solutions of the Fourth Industrial Revolution. In: *[The 2017 Asian Control Conference - ASCC 2017, 2017-12-17 - 2017-12-20, Gold Coast, Australia]*, 2017 2.5, 2.1.1, 2.1.3, 2.2.2, 4.1, 6.1, 6.2.2, 8

[YGE13] Yu, Liyong ; Grüner, Sten ; Epple, Ulrich: An engineerable procedure description method for industrial automation. In: *IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2013, S. 1–8 2.1.3, 4.4.1, 6.1

[YQE10] Yu, Liyong ; Quirós, Gustavo ; Epple, Ulrich: Service-Oriented Process Control for Complex Multifunctional Plants: Concept and Case Study. In: *ETFA 2010: 15th IEEE International Conference on Emerging Technologies and Factory Automation*. Bilbao : IEEE, 2010 4.1

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 978-3-18-526608-9