Evolutionsunterstützung in cyber-physischen Systemen

Dissertation

zur Erlangung des akademischen Grades
Dr. rer. nat.
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg

an der Universität Hamburg eingereichte Dissertation von
Christopher Haubeck
Fachbereich Informatik, VSYS

Hamburg, 12. März 2019

Erstgutachter: Prof. Dr. Winfried Lamersdorf
Zweitgutachter: Prof. Dr.-Ing. Alexander Fay
Drittgutachter: Prof. Dr. Wilhelm Hasselbring
Vorsitzender der Prüfungskommission: Prof. Dr.-Ing Norbert Ritter
Stellv. Vorsitzender der Prüfungskommission: Prof. Dr. Mathias Fischer

Tag der Disputation: 21. Oktober 2019

We don't make mistakes, just happy little accidents.

Bob Ross

Zusammenfassung

Heutige Softwaresysteme sind stark durch einen kontinuierlichen Wandel ihrer zugrundeliegenden Anforderungen und ihrer Umgebung beeinflusst. Um diesem stetigen Veränderungsprozess standzuhalten, müssen Softwaresysteme stetig evolviert werden. Dies gilt insbesondere für cyber-physische Systeme, die durch ihre enge Verknüpfung von physischer und virtueller Welt lange und divergente Lebenszyklen besitzen. Aufgrund von Zeit- oder Kostendruck wird in der Praxis allerdings oft eine unstrukturierte Evolution vorgenommen, die sich in veralteten und fehlenden Spezifikationen manifestiert. Entsprechend fehlt es an einem kohärenten Ansatz, der Wissensartefakte kontinuierlich anpasst und die Fortführung der Evolution unterstützt.

Diese Arbeit leistet einen Beitrag für einen solchen Ansatz indem erstens eine ganzheitliche Beschreibung von Änderungen, zweitens ein softwaretechnischer Prozess zur kontinuierlichen Koevolution von Wissensartefakten und drittens eine kooperative Unterstützung durch einen blockchain-basierten Austausch von Modelldifferenzen entworfen wird. Dazu wird den cyber-physischen Systemen inhärentes Wissen extrahiert, dieses in expliziten Laufzeitartefakten verwaltet und für eine zukünftige Evolution unterstützend eingesetzt. Folgende drei wissenschaftliche Beiträge zur Unterstützung eines derartigen Ansatzes werden im Rahmen dieser Arbeit erzielt:

1. Zunächst wird ein semi-formales Konzept vorgeschlagen, das Evolution in softwaretechnisch abbildbare Schritte unterteilt und die unterschiedlichen Aspekte eines Evolutionsprozesses zusammenhängend beschreibt. Dazu wird jeder Evolutionsschritt mit einer anwendbaren asymmetrischen Modelldifferenz sowie daraus abgeleiteten Gründen und Auswirkungen beschrieben. Dies ermöglicht, ergänzend zu bisherigen Ansätzen, die unterschiedlichen Aspekte einer Evolution zu operationalisieren, maschinenlesbar zu beschreiben und deren Kontext für spezifische Systeme über Domänenmodelle anzupassen.
2. Weiterführend wir durch die Komposition von modellbasierten Methoden ein Ansatz entwickelt, der die Wissenslücke zwischen System und Spezifikation durch eine (halbautomatisierte) Koevolution von deskriptiven Modellartefakten schließt und diese in einer Systemarchitektur für cyber-physische Systeme etabliert. Dafür wird ein autonom ausgeführter, dienstorientierter Prozess vorgeschlagen, der gekapselte Ereignismodelle mittels Laufzeitkomponenten in einem neuartigen, softwaretechnischen Konzept mit beobachteten Zuständen verknüpft.
3. Schließlich wird darauf aufbauend für das Netzwerk von horizontal integrierten Komponenten ein Ansatz ausgearbeitet, der erstmals das Potential eines solchen Cyber-Netzwerks für die Unterstützung von Evolution nutzbar macht. Dafür werden aktuelle Technologien der Modelltransformation und von Peer-to-Peer-Netzwerken verwendet, um über einen ähnlichkeitsbasierten Vergleich über Evolutionsschritte präskriptive Modelle abzuleiten. Dies ermöglicht es dann auch, dem Nutzer über das Cyber-Netzwerk jeweils auf sein aktuelles cyber-physisches System zugeschnittene Handlungsalternativen aufzuzeigen, die konkrete Hilfestellungen für Änderungen und deren Auswirkungen geben.

Abschließend konnte durch die Verwendung von zwei in der Evolutionsforschung etablierten Fallstudien gezeigt werden, dass unterschiedliche Aspekte einer Änderung in einer einheitlichen Beschreibungsform zusammengefasst werden können. Darüber hinaus konnte mit der Einbindung von Entscheidungen des Nutzers und vorhandener modellbasierter Verfahren eine kontinuierliche Koevolution zwischen System und Spezifikation erfolgreich evaluiert werden. Die für die Realisierung vorgeschlagene Systemarchitektur hat gezeigt, dass ein Austausch von modellbasierten Evolutionsbeschreibungen auf Ebene des Cyber-Netzwerkes etabliert werden kann und darüber eine sinnvolle Unterstützung der Weiterentwicklung eines cyber-physischen Systems möglich ist.

Abstract

Today's software systems undergo an everlasting process of changes with respect to their requirements and environments. As a result, software systems must be constantly evolved. Based on the respective interconnection between software and hardware components, cyber-physical systems typically have long and diverse life cycles and are therefore highly affected by evolution. However, due to cost and time constraints, evolution is in practice often carried out without a proper engineering process, leading to a lack of up-to-date specifications. At the same time, current research lacks a coherent approach that preserves knowledge artifacts beyond the development phase in order to support an operator in the task of implementing evolutionary changes.

This thesis contributes to such a coherent approach for cyber-physical systems by investigating a holistic description of changes, an autonomously executed process that co-evolves models at runtime, and operational and cooperative support based on model differences and a self-aware cyber-physical network. For this purpose, the inherent knowledge of the underlying cyber-physiscal system is extracted, managed in operationalized artifacts and proactively provided for future evolutionary changes. This results in three main contributions of this thesis:

1. A semi-formal concept of evolution steps is proposed that describes the evolution process. Each such evolution step is characterized by an asymmetric model difference, its triggers, and effects. In addition to comparable approaches, this concept allows to combine, operationalize, and adapt the various aspects of changes.
2. By composing different model-based techniques, an approach is developed that connects the system and the specification in a semi-automated co-evolution process. For this purpose, descriptive event models are encapsulated in service components that access the underlying states of the cyber-physical system using a novel service-oriented concept for evaluating event statements.
3. In order to support evolution, a network of vertically integrated components of the cyber-physical system is exploited. Based on modern technologies of model transformation and peer-to-peer networks, a comparison process is established that derives prescriptive models from evolution steps already performed in similar systems. This provides new possibilities for providing possible future changes to the human operator that are specifically tailored to the current state of the underlying cyber-physical system.

Finally, through the use of two established application case studies for research in evolution, this thesis shows that various aspects of changes can be combined through a holistic concept of evolution steps. In addition, the traditional gap between a system and its specification is narrowed by combining model-based approaches with a human-in-the-loop. The system architecture developed during the evaluation phase of the project underlying this thesis demonstrates that a cyber-network enables the exchange of evolution steps of different nodes of the network in order to provide meaningful model-based recommendations for future evolution. Thus, new concepts and possibilities for supporting constant evolution processes of cyber-physical systems have been successfully proposed, implemented and also experimentally evaluated.

Danksagung

Während meiner Tätigkeit an der Universität und darüber hinaus haben mich eine Vielzahl von Personen bei meiner Promotion begleitet. Ohne diese Personen wäre die vorliegende Dissertation so nicht möglich gewesen.

Mein erster Dank gilt dabei Professor Lamersdorf für die Betreuung meiner Arbeit und seine Unterstützung in Lehre, Forschung und dem universitären Alltag. Professor Fay danke ich für die Betreuung und herzliche Aufnahme als Forschungspartner und seine umfangreichen fachlichen und methodischen Ratschläge. Weiterhin möchte ich Professor Hasselbring für seine Bereitschaft zur Übernahme eines Gutachtens für diese Arbeit danken.

Als engster und bedeutendster Forschungskollege und Freund möchte ich Jan Ladiges für die vielen Tage mit unzähligen Diskussionen, Erfolgen und Fehlschlägen danken. Im Rahmen unserer engen Zusammenarbeit haben wir gemeinsam die Welt der Wissenschaft kennengelernt und sind an unseren Erfahrungen gewachsen. Dabei hat er durch sein fachliches und methodisches Praxiswissen, seine strukturierte Arbeitsweise und Arbeitsmoral zentrale Ideen dieser Arbeit mitgeprägt. Des weiteren möchte ich mich bei Abhishek Chakraborty und Ireneus Wior für ihre Zusammenarbeit zu unterschiedlichen Zeitpunkten des Forschungsprojektes LinkedFYPA^{2}C danken.

Daneben gab es eine Vielzahl weiterer Kollegen, die mich bei meiner Forschung und täglichen Arbeit unterstützt haben. In erster Linie sind dort Fabian, Dirk, Gabriel, Ante, Volker, Kristof, Heiko, Julian, Wolf, Kai, Lars und Alex zu nennen, mit denen ich zusammen am Arbeitsbereich VSIS arbeiten durfte. mein danke gilt auch den Studierenden Nour-Eddine, Kim, Florian und Paul deren Ergebnisse in Teilen in diese Arbeiten eingeflossen sind. Mein besonderer Dank gilt dabei Anne für ihre Unterstützung und ihren Beitrag zu einer schönen und meist sorgenfreien täglichen Arbeit. Durch alle diese Kollegen konnte ich die Zeit an der Universität über viele Jahre genießen und habe die Zeit an der Universität auch gerne bis zum Äußersten ausgeschöpft.

Ich durfte meine Forschungstätigkeit im Schwerpunktprogramm Design for Future der deutschen Forschungsgemeinschaft durchführen. Auf diesem Weg habe ich viele Erfahrungen und interessante wissenschaftliche Kooperationen durchführen können. Unter vielen Kollegen gilt mein besonders Dank dabei Professor Vogel-Heusser, Sasha Litty, Christopher Pietsch, Christoph Legat, Timo Kehrer, Esteban Arroyo und Jens Folmer. Für Ihre Unterstützung auf den letzten Zügen dieser Arbeit möchte ich zudem Sonja Zaplata, André Basten und meinen weiteren Kollegen der Justizbehörde danken, die mir die Fertigstellung meiner Arbeit ermöglicht haben.

Neben meinen Kollegen gilt mein besonderer Dank meinen Eltern Claudia und Johannes, die mich bei meinem Weg durch das Studium und die anschließende Promotion unterstützt haben. Mit meinen Brüdern Stefan und Daniel sowie meiner weiteren Familie und Freunden möchte ich mich auch herzlich bei allen Personen bedanken, die mich während dieser Zeit begleitet haben und mir andere Perspektiven auf das Leben eröffnet haben. Meine Liebe gilt meiner Verlobten Silvia, die mich insbesondere in der letzten schweren Zeit der Fertigstellung dieser Promotion unterstützt und ermutigt hat.

Hamburg im März 2019
Christopher Haubeck

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Jedes Lebewesen, Roboter oder Gegenstand steht und handelt nicht nur für sich selbst, sondern ist in seine, sich kontinuierlich verändernde, Umgebung eingebettet [Daw06, Dav17]. Dies gilt auch für Softwaresysteme und deren stetigen Wandel in ihrer jeweiligen Umgebung [Leh79, MD08, SS13, GRG+15]. Dabei werden Softwaresysteme in einer technischen Infrastruktur ausgeführt und gleichzeitig durch menschliche Benutzer genutzt und bewertet [GG08, dSQTR07]. Entsprechend unterliegen Softwaresysteme nicht nur sich verändernden technischen Anforderungen, wie neuen Funktionalitäten, sondern auch sich stetig verändernden Nutzungsanforderungen [GG08].

Alterung von Software

Durch diesen kontinuierlichen Änderungsprozess entsteht eine Alterung von Softwaresystemen [Leh79] - wobei Software als immaterielles Gut nicht im biologischen Sinne durch die Degeneration seiner Bestandteile altert, sondern vielmehr im Vergleich zu seiner fortschreitenden Umgebung degeneriert [HWB+13]. Eine derartige Alterung umfasst insbesondere unklare oder veraltete Systemarchitekturen, ungeplante und unstrukturierte Entwicklungen, als auch den Wissensverlust aufgrund von fehlender Dokumentation oder Mitarbeiterfluktuation [SDK+11].

Entsprechend hielt Lehmann [Leh79, Leh96] bereits in den ersten Arbeiten zur Softwareevolution fest, dass Softwaresysteme kontinuierlich evolviert werden müssen, um ihrer Degeneration entgegenzuwirken. In dem dadurch entstehenden, kontinuierlichen Evolutionsprozess eines operativen Softwaresystems werden laufend Änderungen durchgeführt, um das System seiner veränderten Umgebung anzupassen und die Qualität der Software zu erhalten [RB02]. Dabei nimmt die Komplexität und die Frequenz im Evolutionsprozess stetig zu [Leh79, Hat07]. Laut einer Studie von Les Hatton [Hat07] steigt der jährliche Umfang eines Softwaresystems durch deren Evolution durchschnittlich um 10-16 % und Amazon gibt beispielsweise an, dass im Durchschnitt bereits alle elf Sekunden eine neue Version eines ihrer Dienste veröffentlicht wird [Jen11]. Da sich der Nutzen einer Software stets an der Erfüllung von Anforderungen seiner Umgebungen orientiert [GG08], verschiebt sich dadurch die Wertschöpfung für Softwareprodukte stetig von der initialen Entwicklung hin zu späteren Phasen des Softwarelebenszyklus [Som12]. Hierbei ist festzuhalten, dass eine der größten Herausforderungen darin besteht, dieser steigenden Frequenz und Bedeutung von Softwareevolution durch entsprechende Methoden zu begegnen [GRG+15].

Wissensmangel bei Änderungen

Um mit dieser notwendigen Evolution umgehen zu können, bedarf es bei der Anpassung von Softwaresystemen eines entsprechenden Verständnisses und Wissens über das System und seine Umgebung [SS13]. Denn fehlendes Wissen ist ein Hauptgrund für das Scheitern von Evolutionsprojekten [SS13]. Eines der spektakulärsten und prominentesten Beispiele hierfür ist der fehlgeschlagene Jungfernflug der Ariane 5 Rakete (siehe dazu [Dow97]). Der Absturz wurde

zu einem guten Teil darauf zurückgeführt, dass eine Steuerungssoftware aus einer alten Version der Rakete verwendet wurde. Diese beruhte allerdings auf alten Flugbahnberechnungen, welche zwar im Softwarecode versteckt, aber nicht explizit bei der Weiterentwicklung vorlagen. Durch dieses fehlende Wissen kam es schlussendlich in einer Kettenreaktion zur Selbstzerstörung der Rakete und zum Scheitern der Mission.

Das Beispiel zeigt, dass Änderungen inhärent komplex und schwer durchzuführen sind, sowie ein hohes Maß an potenziellen Fehlern beinhalten. Dabei sollten Änderungen, wie bei der Entwicklung von Software durch entsprechende Entwicklungsmodelle vorgegeben, auch während der Evolution in einem klar definierten Prozess durchgeführt werden [SS13]. Dementsprechend sollte das Wissen, über den Quellcode hinausgehend, in einer strukturierten Form abgelegt werden. Dies ist aber in der Praxis nur selten gegeben [VHFF+15, HWB+13]. Der Grund dafür ist, dass eine Vielzahl von nicht im Vorfeld antizipierter Änderungen erst zur Laufzeit am operativen System selbst durchgeführt werden [VHDF+14]. Diese Ad-hoc-Änderungen werden als Reaktion auf eine veränderte Umgebung ohne eine entsprechende methodische Unterstützung, eine ausreichende Dokumentation oder Qualitätskontrolle durchgeführt [GRG+15, HWB+13, SS13]. Dies umfasst auch, dass in der Regel keine geeignete softwaretechnische Evolutionsunterstützung vorhanden ist, welche nach Ducasse und Pollet [DP09] die Dokumentation, die Analyse, die Koevolution zwischen zwei verschiedenen Artefakten und die Fortführung der Evolution zum Ziel hat.

Der damit einhergehende Verlust von Wissen, sowohl durch Evolution als auch Erneuerung von Systemen, wurde in der Softwaretechnik bereits früh aufgezeigt (vergleiche [Par94]). Entsprechend bleibt festzustellen, dass für die Evolution in der Regel keine aktuellen Wissensartefakte des Systemverhaltens vorliegen [SS13, Lad18] und dieser Mangel an Wissen einer effizienten und effektiven Evolution entgegensteht [GRG+15].

Koevolution von System und Spezifikation

Wie es bereits Lehmann [Leh79, Leh96] als ein Gesetz der Evolution formuliert, führt Wissensmangel dazu, dass die Qualität der Software während ihres Lebenszyklus stetig abnimmt. Die unterschiedliche Geschwindigkeit bei der Evolution von Spezifikation und Implementierung des Systems erfordert eine Synchronisierung dieser Evolution [DP09] in Form einer gemeinsamen Koevolution. Diese Koevolution von Spezifikation und System umfasst neben der Konsistenz von technischen und benutzereigenen Anforderungen auch die allgemeine Softwarequalität, wie ihre Funktionalität, Performanz, Zuverlässigkeit und Wartbarkeit [GRG+15]. Dabei wird der Begriff der Softwarequalität bezüglich Evolution zu oft auf die Fehlerfreiheit und Performanz reduziert, was damit zusammenhängt, dass weitere Qualitätseigenschaften nur schwer messbar sind [SS13]. Die Notwendigkeit, die Qualität der Software und ihrer Dokumentation messbar zu erfassen, um die Wartbarkeit und Wiederverwendbarkeit zu gewährleisten, gilt deshalb als zentrale Herausforderung der Evolution [SS13].

Die Messbarkeit umfasst dabei nicht nur vergangene und momentane Zustände des Systems, sondern schließt bei einer vollständigen Evolution auch explizit zukünftige Zustände mit ein. In diesem Zusammenhang wird bei einem modellbasierten Vorgehen, als eine Möglichkeit die Komplexität von Evolution zu beherrschen, von deskriptiven, momentanen Beschreibungen und präskriptiven, zukünftigen Beschreibungen gesprochen [Keh15, Küh06]. Zur Messbarkeit der Evolution sollten deskriptive Beschreibungen automatisch aus vorhandenen Informationsquellen extrahiert werden, um diese mit möglichen präskriptiven Beschreibungen abzugleichen. Denn dadurch kann das System stetig im Einklang mit den Anforderungen seiner Spezifikation gehalten werden. Eine Evolutionsunterstützung inkludiert deshalb insbesondere Methoden, die

helfen, die durch den Wissensverlust hervorgerufene Divergenz zwischen dem laufenden System und seiner Spezifikation durch eine Koevolution zu verringern [DP09, HWB+13, VHFST15].

Dabei sollte auch berücksichtigt werden, dass die Qualität einer Evolution nur im Zusammenspiel mit dem Nutzer des Systems bewertet werden kann [LHFL14b]. Hierbei gilt es, das Spannungsfeld zwischen der Qualität einer Spezifikation und der dafür notwendigen Arbeitsleistung des Nutzers durch ein geeignetes Vorgehen zu gewährleisten (vergleiche [MKCC03]). Entsprechend sollte der manuelle Aufwand für eine Koevolution von Spezifikation und System möglichst gering gehalten werden [LHFL14a].

Cyber-physische Systeme als Anwendungsgebiet für Evolution

Nicht jedes System ist von der Alterung und dem damit einhergehenden Qualitätsverlust gleichermaßen betroffen, denn diese tritt insbesondere in Systemen mit langen und divergierenden Lebenszyklen auf [GRG+15, VHFF+15]. Cyber-physische Systeme repräsentieren eine Gruppe von Systemen, die solche langen und divergierenden Lebenszyklen besitzen und deshalb von Alterung besonders betroffen sind [VHFF+15].

Dabei ist ein cyber-physisches System eine Verknüpfung von realen (physischen) Komponenten und softwaretechnischen (virtuellen) Komponente, die über ein Informationsnetzwerk miteinander kommunizieren [GB12]. Ein zentraler Aspekt welcher cyber-physische Systeme dabei von anderen Systemen unterscheidet, ist deshalb ihre inhärente Interdisziplinarität [DLV12]. Denn die Kernidee von cyber-physische Systemen[1] ist die Verknüpfung von konzeptionellen und technischen Aspekten der Interaktion von physischer und virtueller Welt [VHHC+17]. Durch diese starke Verflechtung der verschiedenen Disziplinen, wie Physik, Elektrik und Software, muss Wissen der Evolution deshalb in cyber-physischen Systemen immer interdisziplinär betrachtet werden [VHFST15]. Für die Evolution liegt deshalb eine besondere Schwierigkeit in der Erfassung dieser interdisziplinären Dynamiken [AAH+11]. Diesbezüglich haben Expertenbefragungen im Bereich der Produktionsautomatisierungssysteme, als ein Anwendungsfeld von cyber-physische Systemen, gezeigt, dass ein hoher Bedarf an strukturieren und systematischen Unterstützungsmethoden besteht [BS09]. Entsprechend sollten cyber-physische Systeme durch Methoden und intelligente sowie smarte Architekturen systematisch unterstützt werden [ZT16].

Die inhärente Durchdringung des Internets in cyber-physischen Systemen ermöglicht es, Evolution in verteilten Netzwerken zu erfassen. Dadurch werden vergleichende Verfahren ermöglicht, die Relationen zwischen der Umgebung, den Anforderungen und Zuständen von vernetzten Systemen aufzeigen können [LBK15]. Nach Lee et al. [LKY14] wird dabei mit steigender Verbreitung von cyber-physischen Systemen und Cloud Computing die zukünftige Industrie in der Lage sein, durch ihr umfassendes Informationssystem ein Bewusstsein von sich selbst zu schaffen. Ein sich selbst bewusstes System kann dadurch seinen Zustand und dessen Verschlechterung selbst beurteilen und Informationen aus vergleichbaren Systemen für präskriptive Vorschläge und intelligente Entscheidungen nutzen, um mögliche Probleme aktiv zu vermeiden [LKY14]. Für eine solche kollaborative Zusammenarbeit ist eine gemeinschaftliche Sammlung und Verarbeitung von Artefakten notwendig [GG08]. Denn dadurch könnte nicht nur der Zustand, sondern auch dessen Evolution ganzheitlich betrachtet werden.

[1] Nachfolgend wird unter dem Begriff des Systems auch cyber-physische Systeme inkludiert. Wenn eine Betonung auf der Interaktion zwischen Software und Hardware liegt, wird stets der Begriff cyber-physisches System verwendet.

1.2 Problemstellungen

Zusammenfassend ergeben sich folgende **Problemstellungen**, welche in dieser Arbeit betrachtet werden:

Evolutionsartefakt: Im Rahmen der Entwicklung von Systemen entstehen durch die Leistung und Erfahrung der Entwickler und Nutzer Wissensartefakte. Diese müssen zum einen über die Entwicklungsphase hinaus als Evolutionsartefakte aktuell gehalten und zum anderen mit Laufzeitinformationen verknüpft werden. Dazu bedarf es einer Koevolution von Spezifikation und Systemverhalten, um Evolution zu verstehen und zu planen (vergleiche [DP09, VHFF+15, GRG+15]).

Kohärente Unterstützung: Es bedarf einer kohärenten Methode und Architektur, um die verschiedenen vorhanden, insbesondere modellbasierten Ansätze der Softwaretechnik zu verknüpfen. Dadurch soll Evolutionswissen für die Dokumentation, Analyse, Koevolution und Evolutionsunterstützung besser bewahrt und verwendet werden (vergleiche [GRG+15, Keh15, DP09]).

Selbstwahrnehmung: Insbesondere unter der Bedingung von kurzen Änderungszyklen müssen cyber-physische Systeme eine Selbstwahrnehmung entwickeln. Dieses umfasst, dass Evolution automatisiert erfasst und maschinenlesbar unterstützt wird. Dabei sollten Methoden und smarte Architekturen mit geringem manuellem Aufwand das cyber-physische System unterstützen, um den Trade-off zwischen der Qualität von Evolutionsartefakten und der Produktivität des Nutzers zu verbessern (vergleiche [ZT16, MKCC03]).

Fortführung: Cyber-physische Systeme sollten ihre Vernetzung nutzen, um in einem kooperativen Prozess eine ganzheitliche Betrachtung, nicht nur ihres Zustands, sondern auch ihrer Evolution zu ermöglichen. Dazu sollte das implizite Verhalten des Systems und vergleichbarer Systeme zur Laufzeit explizit gemacht werden. Dieses sollte deskriptiv bei der Evolution zur Verfügung gestellt werden, um beispielsweise präskriptive Beschreibungen für die Fortführung von Evolution abzuleiten (vergleiche [LKY14, LBK15, GG08, OSL11]).

Rahmenbedingung: Evolutionsunterstützung in cyber-physischen Systemen bedarf der Berücksichtigung ihrer Rahmenbedingungen, wie der Interdisziplinarität, der Einbindung von Legacy-Systemen, der erforderlichen minimalen Informationsmodellierung und der Realzeitanforderungen auf den niedrigen funktionalen Ebenen (vergleiche [Lee08, VHFST15, DLV12]).

1.3 Themenfeld der Arbeit

Im folgenden Abschnitt wird das Themenfeld der Arbeit genauer definiert, indem die Problemstellungen durch Forschungsfragen konkretisiert und die Forschungsgegenstände zur Beantwortung dieser Fragestellungen vorgestellt werden.

Forschungsfragen

Die Hauptforschungsfrage beschreibt den aus den Problemstellungen hervorgehenden Bedarf einer kohärenten Evolutionsunterstützung über Wissensartefakte in einer selbstwahrnehmenden Systemarchitektur für cyber-physische Systeme:

Hauptforschungsfrage: Mit welcher Methode und unter welcher Systemarchitektur können modellbasierte Evolutionsartefakte eines cyber-physischen Systems systematisch in Versionen koevolviert und in Varianten fortgeführt werden?

Die Hauptforschungsfrage umfasst damit zwei Aspekte der Softwaretechnik. Zum einen den methodischen Aspekt, eine geeignete Methode zu entwerfen, und zum anderen den operationalen Aspekt, eines dazu passenden Architekturmodells. Ziel dabei ist die systematische Unterstützung und Fortführung von Evolution in cyber-physischen Systemen. Entsprechend gliedert sich die vorliegende Arbeit in zwei miteinander verzahnte Teile. Der erste Teil fokussiert auf die Forschungsfrage der *Methodik*:

Methodik: Wie kann eine Methode definiert werden, in der Nutzer beim Verständnis, der Erhaltung und der Fortführung von Evolution systematisch durch Evolutionsartefakte unterstützt werden?

In dieser Forschungsfrage werden die drei Problemstellungen der Evolutionsartefakte, Selbstwahrnehmung und Fortführung thematisiert. Das Ziel ist der, als Problem identifizierte Entwurf einer kohärenten Methode, um Wissen zu bewahren. Der zweite Teil der Arbeit behandelt die Forschungsfrage der *Operationalität*:

Operationalität: Welche Systemarchitektur ist in einer cyber-physischen Umgebung geeignet, Evolutionsartefakte zu extrahieren und in Softwareeinheiten zu verarbeiten, um eine Selbstwahrnehmung von Evolution zu erreichen?

Der operationale Aspekt fokussiert auf die Problemstellungen von cyber-physischen Systemen indem Evolutionsartefakte genutzt werden, um eine geeignete Evolutionsumgebung zu schaffen. Diese hat zum Ziel, die identifizierte Selbstwahrnehmung und Fortführung von Evolution zu ermöglichen.

Neben der Aspektdimension des methodischen und operationalen Aspekts, unterscheidet die anfängliche Forschungsfrage mit Versionen und Varianten in einer Verknüpfungsdimension. Dabei wird unterschieden, dass Evolutionsartefakte sowohl zeitlich als auch räumlich miteinander verknüpft werden können. Hierfür wirft die Arbeit zunächst eine allgemeine Fragestellung auf, mit der die zugrundeliegende Evolution und deren Unterstützung charakterisiert wird:

Evolutionsverständnis: Was ist eine geeignete Beschreibung von Evolution, um Evolutionsartefakte als zentrale methodische Einheiten zur Laufzeit zu etablieren?

Die *allgemeine* Verknüpfung fokussiert insbesondere die Problemstellung der kohärenten Unterstützung und geeigneten Systemarchitektur, indem als Ziel eine geeignete Beschreibung in Evolutionsartefakten avisiert wird.

Unter der allgemeinen Fragestellung betrachtet die *zeitliche* Verknüpfung das Evolutionswissen in unterschiedlichen Versionen:

Temporalität: Wie kann Evolutionswissen aus unterschiedlichen Artefakten der Evolutionshistorie eines cyber-physischen Systems systematisch gewonnen, verwaltet und zur laufenden Evolution konsistent gehalten werden, um Evolution durch historische Erfahrungen zu unterstützen?

Die Forschungsfrage widmet sich insbesondere der Problemstellung von Evolutionsartefakten und der damit einhergehenden Kohärenz von methodischen als auch operationalen Aspekten. Ziel ist die Selbstwahrnehmung unter den Rahmenbedingungen von heterogenen cyber-physischen Systemen zu verbessern.

Bei der *räumlichen* Verknüpfung werden Varianten von Komponenten in der Forschungsfrage der Verteilung betrachtet:

Verteilung: Inwieweit können systematisch erstellte Evolutionsartefakte in horizontal integrierten Komponenten verbreitet und genutzt werden, um die Evolution zu unterstützen?

Als Ziel dieser Fragestellung wird die Problemstellung der Fortführung von Evolution adressiert, indem aufbauend auf der Selbstwahrnehmung von cyber-physischen Systemen Evolutionsartefakte systemübergreifend genutzt werden.

Forschungsgegenstände

Wie die Forschungsfragen beantwortet werden sollen, drückt sich in den Forschungsgegenständen der Arbeit aus. Diese sind durch eine Aspekt $\times$ Verknüpfungs-Matrix in Tabelle 1.1 zusammenfassend beschrieben. Dazu werden die methodischen und operativen Aspekte mit der allgemeinen, zeitlichen und räumlichen Verknüpfung in Verbindung gebracht, um dann jeweils die Forschungsgegenstände für diese Kreuzverbindungen aufzuzeigen.

	methodischer Aspekt	**operationaler Aspekt**
allgemeine Verknüpfung	Schrittbasiertes Evolutionskonzept	Cyber-physische Systeme
zeitliche Verknüpfung	Prozess der Evolutionsunterstützung	Wissenstragende Komponente
räumliche Verknüpfung	Schrittbasierter Vergleichsprozess	Marktplatz für Evolutionsschritte

Tabelle 1.1: Forschungsgegenstände der Arbeit dargestellt als Aspekt $\times$ Verknüpfungs-Matrix

Um eine *allgemeine* Verknüpfung herzustellen, wird in dieser Arbeit sowohl ein methodisches als auch ein operationales Konzept entworfen. Dazu wird ein *methodisches* Evolutionsverständnis vorgeschlagen, unter dem Softwareevolution operationalisiert beschrieben werden kann. Diese Beschreibung verwendet eine Darstellung in Evolutionsschritten, die das zentrale Evolutionsartefakt dieser Arbeit darstellen. Welche funktionalen Ebenen dafür vorhanden sein müssen und wie die Rahmenbedingungen von cyber-physischen Systemen dafür spezifiziert werden, wird dabei *operativ* untersucht.

Die *zeitliche* Verknüpfung betrachtet *methodisch* einen Evolutionsunterstützungsprozess, der modellbasierte Evolutionsartefakte aus unterschiedlichen Informationsquellen generieren kann und diese systematisch in einem Prozess koevolviert. Aus diesen koevolvierenden Evolutionsartefakten werden dann Evolutionsschritte als deskriptive Beschreibung der dem

System inhärenten Evolution extrahiert. Der *operative* Aspekt entwirft eine Systemarchitektur für unterschiedliche cyber-physische Systeme, um eine wissenstragende Komponente zu etablieren, die automatisiert den Evolutionsunterstützungsprozess unter Verwendung passender Softwarekonzepte realisiert.

Die *räumliche* Verknüpfung untersucht die Fortführung von Evolution, indem *methodisch* ein systemübergreifender Vergleichsprozess mit Evolutionsschritten entwickelt wird. Dieser ermöglicht es, Evolutionsprozesse abzubilden und den Nutzer durch modellbasierte, präskriptive Vorschläge zu unterstützen. Die dafür notwendigen horizontal integrierten Komponenten eines cyber-physischen Systems werden *operativ* durch einen Marktplatz für Evolutionsschritte realisiert, der auf Basis einer Peer-to-Peer Blockchain den Austausch von Evolutionsschritten ermöglicht.

1.4 Aufbau der Arbeit

In Kapitel 2 werden die technischen Grundlagen der Arbeit erläutert. Dazu wird zunächst der Begriff der Evolution mit seinen Gründen, Auswirkungen und Klassifizierungen betrachtet. Weiterhin werden wesentliche methodische und softwaretechnische Ansätze, die im Rahmen dieser Arbeit zur Evolutionsunterstützung tangiert werden, erläutert. Abgeschlossen wird die Darstellung der Grundlagen mit der Vorstellung von cyber-physischen Systemen in den Anwendungsgebieten der Produktionssysteme und der Informationssysteme.

Der Aufbau der folgenden zwei Kapitel folgt den identifizierten Forschungsfragen. In Kapitel 3 wird dementsprechend die methodische Fragestellung erörtert und in Kapitel 4 die operative Fragestellung. Zur Beantwortung der Fragestellungen werden Hypothesen für die betrachteten Forschungsgegenstände aufgestellt, die entlang der Verknüpfungen anschließend diskutiert werden. Aus methodischer Sicht sind dies die Konzeption von Evolutionsschritten, der Prozess der Verarbeitung von Informationen aus heterogenen Quellartefakten, die Abstraktion in modellbasierten Evolutionsschritten und der Ansatz zur Dokumentation und Fortführung eines Evolutionsprozesses. Das operative Kapitel befasst sich mit den funktionalen Ebenen und der Spezifikation von cyber-physischen Systemen, der Systemarchitektur für eine wissenstragende Komponente und dem Aufbau eines Marktplatzes auf Basis eines cyber-physischen Evolutionsnetzwerks.

Im darauf folgenden Kapitel 5 werden vergleichbare Arbeiten vorgestellt und mit dem in dieser Arbeit entwickelten Ansatz verglichen. Dies umfasst Ansätze zur Beschreibung von Evolution, Methoden und Systemarchitekturen zur Koevolution und Verfahren zur Evolutionsunterstützung von cyber-physischen Systemen.

Eine Evaluation der vorgeschlagenen Konzepte wird in Kapitel 6 vorgestellt. Dazu werden zunächst Ziele der Evaluation definiert, die dann jeweils durch Experimente überprüft werden. Für die einzelnen Experimente werden dabei jeweils der Experimentaufbau, ein Proof-of-Concept für das Verständnis der Lösung und eine Diskussion der Ergebnisse vorgestellt. Als Fallstudien werden zwei unterschiedliche cyber-physische Systeme im Bereich der Produktionssysteme und Informationssysteme vorgestellt und verwendet.

Das abschließende Kapitel 7 gibt ein Fazit der Arbeit. Dies umfasst die inhaltliche Zusammenfassung der Ergebnisse und die Betrachtung des wissenschaftlichen Beitrags. In einem Ausblick werden mögliche zukünftige Arbeiten betrachtet, die an diese Arbeit anschließen können.

2 Grundlagen der Evolution

Das folgende Kapitel führt in die technischen Grundlagen ein, die für das Verständnis dieser Arbeit notwendig sind. In Abschnitt 2.1 wird dazu zunächst der Begriff der Softwareevolution allgemein erörtert. Anschließend wird die Evolutionsunterstützung in Abschnitt 2.2 erörtert. Abschnitt 2.3 stellt Softwarekonzepte vor, die im Rahmen dieser Arbeit Anwendung finden. Abschließend werden die Anwendungsgebiete der Arbeit in Abschnitt 2.4 eingeführt.

2.1 Evolution von Systemen

Evolution beschreibt den Änderungsprozess eines Systems. Dies wird zunächst anhand des Evolutionsbegriffes und dann durch typische Gründe und Auswirkungen erläutert. Anschließend werden unterschiedliche Aspekte der Evolution vorgestellt, deren Bezüge zur Arbeit dabei jeweils kurz aufgezeigt werden.

2.1.1 Begriff der Evolution

Obwohl eine **Evolution**[1] von Softwaresystemen schon immer stattfand, gehen die ersten fundamentalen Arbeiten dazu auf Lehman [Leh79, Leh96] zurück. In diesen werden anhand von empirischen Studien Evolutionsgesetze (*laws of evolution*) formuliert. Diese auf Beobachtungen basierenden Gesetzmäßigkeiten besagen, dass ein System kontinuierlichen Änderungen unter einem begrenzten Wachstum unterliegt, dabei die Komplexität stetig zunimmt und die Qualität, sowie die Produktivität und Effektivität abnimmt (vergleiche [Leh79, Leh96]).

Dennoch ist bis heute Evolution von Software, insbesondere in Abgrenzung zu deren **Wartung** (*maintenance*), nicht allgemeingültig definiert [BR00, TRdL07]. Einige Autoren trennen klar zwischen Wartung und Evolution. So definieren Tripathy und Naik [TN14] Wartung als geplante und vorbeugende Maßnahme, um die vorhandene und gewünschte Funktionalität eines Systems zu erhalten. Unter dieser Sichtweise beschreibt die Evolution eine kontinuierliche Veränderung von einem niedrigen, schlechteren Zustand in einen höheren, besseren Zustand [Art88]. Andere Autoren sehen Wartung dabei auch als einen möglichen Auslöser und Bestandteil der Evolution [SDK+11].

Unter Berücksichtigung von iterativen Entwicklungsverfahren, bei denen Softwaresysteme in einer wiederkehrenden Schleife die Phasen der Anforderungsanalyse, des Designs, der Implementierung, des Testens, sowie der Abnahme durchlaufen, lassen sich auch Entwicklung und Evolution nicht mehr klar voneinander trennen [SS13]. Denn wenn Funktionalität in kleine, iterativ bearbeitete Teile aufgeteilt wird, hat der Entwickler die Möglichkeit von vorangegangenen Iterationen zu lernen [BT75]. Deutlich zeigt sich dies im so genannten *DevOps*-Ansatz. Dabei sollen die Phasen der Entwicklung und des Betriebs von Software sowohl organisatorisch und methodisch, als auch technologisch miteinander verknüpft werden und in eine kontinuierliche Iteration übergehen [FS14]. Dies hat zur Folge, dass die klassische Trennung zwischen Entwicklung und Evolution, die mit dem ersten produktiven Einsatz des Systems beginnt, aufgeweicht wird [SS13].

[1] Begriffe die im späteren Verlauf der Arbeit eine wichtige Rolle spielen, sind im Grundlagenkapitel fett markiert.

Einige Autoren sehen Evolution rein auf Software oder auch Softwarecode bezogen, während beispielsweise Torchiano [TRdL07] explizit auch technische und betriebswirtschaftliche Aktivitäten in die Evolution einbeziehen. In diesem Zusammenhang sind auch cyber-physische Systeme zu sehen, die Evolution nicht nur bezüglich Software betrachten, sondern Softwaresysteme explizit mit der Hardware und der damit verbunden Elektrik und Mechanik zusammen betrachten.

Ein daran anschließender Trend in der Softwareentwicklung ist, dass Softwaresysteme nicht mehr nur auf Grundlage des Codes betrachtet wird, sondern unter Einbeziehung aller möglichen **Artefakte** [DGLP08]. Entsprechend ist Evolution auch immer im Kontext der Systemumgebung mit Einbeziehung der verschiedenen Artefakte zu betrachten. Dies sind z.B. Software, Hardware oder die Spezifikation des Systems.

Dabei sind, vergleichbar zur Entwicklung, Modelle mittlerweile als unverzichtbar anzusehen, da diese eine Beschreibung des Systems auf einem hohen Abstraktionsniveau erlauben [BFH+10]. Ein **Modellartefakt** beschreibt dem folgend eine zweckbestimmte Abbildung eines Systems und kann als Ersatz für das eigentliche System verstanden werden [LK07]. Jedes Modell vernachlässigt dabei bestimmte Aspekte des Systems, um die Zusammenhänge im Systemverhalten für den menschlichen Verstand oder ein anderes Softwaresystem verständlicher zu machen. Zur Beschreibung von Evolution können Modelle dabei sowohl deskriptiv, als abstrakte Beschreibung des vorhandenen Systems, oder präskriptiv, als Beschreibung eines zu entwickelnden oder auszuführenden Systems, verwendet werden [Küh06].

2.1.2 Gründe und Auswirkung von Evolution

Softwareevolution kann aus unterschiedlichen Gründen verfolgt werden. Im Allgemeinen kann davon gesprochen werden, dass die Alterung bzw. Degeneration des Systems verhindert oder zumindest verringert werden soll [SDK+11, Par94]. Dabei altern Softwaresysteme nicht im eigentlichen Sinne, sondern im Verhältnis zu den immerwährenden Änderungen seiner Anforderungen, Spezifikation, Nutzung und der unterliegenden Infrastruktur [HWB+13]. Treiber einer solchen Alterung sind beispielsweise neue Anforderungen, wie veränderte Produkte, Nutzerwünsche, neue Geschäftschancen oder auch Veränderungen des unterliegenden Technologie-Stacks [SDK+11, GRG+15].

Die Alterung führt zu einer Reduzierung der Qualität, die für Software nach dem ISO 25000 Standard [Int05] beschrieben werden kann [GRG+15]. Der Standard unterscheidet dabei hinsichtlich der Softwarequalität zwischen **funktionalen** und **nichtfunktionalen Eigenschaften**. Funktionale Eigenschaften beschreiben, ob die Software korrekt funktioniert und die an die Software gestellten funktionalen Anforderungen erfüllt. Nichtfunktionale Eigenschaften geben darüber Auskunft, wie gut diese Anforderungen bezüglich verschiedener Kriterien erfüllt werden. Aus Sicht der Evolution wird in dieser Arbeit insbesondere auf die nichtfunktionalen Eigenschaften der Performanz und Konsistenz fokussiert. Die Performanz beschreibt die Leistung eines zugrundeliegenden Systems. Die Konsistenz beschreibt sowohl die Übereinstimmung von verschiedenen Artefakten eines Systems, als auch die Übereinstimmung von Teilen innerhalb eines Artefakts. Dabei kann auch die Korrektheit des Systems als Konsistenz zwischen dem Verhalten und der Verhaltensspezifikation aufgefasst werden.

Um die Auswirkung von Evolution auf diese Eigenschaften objektiv messen zu können werden **Metriken** oder Kenngrößen verwendet [Kri88]. Metriken dienen der Bewertung von Evolution als Hilfsinstrument, um aus den Eigenschaften quantitative Werte ableiten zu können und diese automatisiert zu verarbeiten. Im Allgemeinen ist eine Metrik somit eine Abbildung von einer Entität auf eine quantitative Größe, die dazu dient, eine bestimmte Eigenschaft der Entität zu

beschreiben [SL10]. Zu diesem Zweck existieren eine Vielzahl von Metrikenkatalogen. Für die Qualität von objektorientierten Softwaresystemen entwickelten beispielsweise Chidamber und Kemerer [CK94] einen Metrikenkatalog und für Fertigungssysteme betrachten Ladiges et al. [LFHL13] Metriken zur Evolutionsbewertung.

Durch die Evolution werden nicht nur stetig Eigenschaften geändert, sondern auch fortwährend neue **Zustände** des Systems und seiner Artefakte erreicht. Dabei ist jeder neue Zustand nicht isoliert zu betrachten, sondern es existieren Überlappungen zu den vorherigen und vergleichbaren Zuständen [TN14]. Zustände eines Systems oder Artefakts werden in ihrer zeitlichen Abfolge als **Versionen** bezeichnet. Entsprechend repräsentiert zwei Versionen dasselbe System oder Artefakt zu zwei unterschiedlichen Zeitpunkten. Weiterhin können in einer Evolution auch **Varianten** vorhanden sein. Varianten können als zwei Version verstanden werden, welche zueinander koexistieren [CW98a]. Diese Varianten repräsentieren dabei Artefakte oder Systeme mit durch den Nutzer unterscheidbaren Eigenschaften. Auch Varianten können Überlappungen aufweisen, da sie häufig von einer ursprünglich gleichen Basis abgeleitet wurden oder ähnliche Änderungen durchlaufen haben.

2.1.3 Klassifizierung von Änderungen

Zur Klassifizierung von Evolution existieren unterschiedliche Ansätze, die sich allerdings teils in Abhängigkeit zur betrachteten Anwendungsdomäne befinden. Als bekannteste Klassifizierung nach Buckley et al. [BMZ+05] und Chapin et al. [CHK+01] können Änderungen anhand der Frageworte *was*, *warum*, *wer*, *wann*, *wo* und *wie* klassifiziert werden. Die Autoren definieren zusätzlich Ausprägungen dieser Aspekte von Änderungen. Während die Ausprägungen in der Literatur unterschiedlich formuliert werden, können die Aspekte als allgemein akzeptiert angesehen werden.

Buckley et al. [BMZ+05] identifizieren vier Aspekte: Zum einen die *Systemeigenschaften* (*system properties* (was)). Diese Dimension beschreibt, welche Eigenschaften des Systems verändert werden. Mit dem *Änderungsobjekt* (*object of change* (wo)) wird das veränderte Artefakt und der auf das Artefakt wirkende Effekt beschrieben. Daneben wird die *zeitliche Eigenschaft* (*temporal properties* (wann)) als Dimension identifiziert. Diese beschreibt wann und in welcher Frequenz eine Änderung stattfindet und ob die Änderung antizipiert wurde. Die Dimension der *Änderungsunterstützung* (*change support* (wie)) beschäftigt sich mit den Mechanismen zur Messung, Analyse, Management, Kontrolle oder Implementierung von Änderungen. Zusätzlich werden zwei weitere Aspekte von Chapin et al. [CHK+01] in einer evidenzbasierten Klassifikation definiert. Eine Dimension ist der *Ursprung von Änderungen* (*activities* (wer)), welcher durch das auslösende Artefakt definiert ist. Weiterhin werden *Evolutionstypen* definiert (*type of evolution* (warum)), mit denen Änderungen in verschiedene Cluster eingeteilt werden, um den Grund der Änderungen zu erfassen.

Das Konzept der Evolutionsschrittartefakte, das diese Arbeit vorschlägt, soll Evolutionswissen von allen Aspekten operationalisiert abbilden, um den Nutzer des Systems bei der Evolution zu unterstützen. Dazu werden die vorgestellten Aspekte von Änderungen in Evolutionsschritten aufgegriffen.

2.2 Evolutionsunterstützung

Als Konsequenz aus den Gesetzmäßigkeiten der Evolution schlussfolgern Sneed et al. [SS13], dass die Evolution geplant und die Qualität der Software laufend untersucht werden muss. Diese **Evolutionsunterstützung** gilt als große Herausforderung der Evolution [GRG+15]. Die dafür notwendigen Aktivitäten und Prozesse werden zusammen mit einer entsprechenden Klassifizierung nachfolgend vorgestellt. Anschließend werden relevante Methoden, die im Rahmen dieser Arbeit verwendet werden, behandelt. Dabei bedarf es für die Evolutionsunterstützung an Methoden, Konzepten und Werkzeugen aus unterschiedlichen softwaretechnischen Forschungsbereichen [GRG+15].

2.2.1 Klassifizierung von Aktivitäten der Evolution

Die zu unterstützenden **Aktivitäten** der Evolution umfassen u.a. die Planung, die Dokumentation, die Analyse und Auslieferung von Änderungen [SS13]. In dieser aktivitätsorientierten Sichtweise wird in der Literatur zusätzlich nach der Art der Unterstützung durch diese Aktivitäten unterteilt. Die dafür gebräuchlichste Klassifizierung orientiert sich an einer Studie von Lientz und Swanson [LST78], welche sich auch im ISO/IEC Standard 12207 [Int95] für Softwarewartung niederschlägt. Danach lassen sich Aktivitäten in korrigierende (*corrective*), verbessernde (*perfective*) und adaptive (*adaptive*) Aktivitäten einteilen. Korrigierende Aktivitäten beseitigen Fehler im System. Verbessernde Aktivitäten realisieren veränderte oder optimierte Anforderungen. Adaptive Aktivitäten beschreiben Anpassungen aufgrund einer veränderten Umgebung. In dieser Arbeit sollen eine Methodik und eine Architektur entwickelt werden, die für alle Arten von Aktivitäten einen Mehrwert bietet.

2.2.2 Iterative Softwareentwicklung

Als Folge der iterativen Softwareentwicklung wird in vielen Bereichen eine wiederkehrende Entwicklung zur fortlaufenden Integration von neuen Versionen angestrebt. Dabei werden verschiedene Phasen der Softwareentwicklung aufgegriffen und wiederkehrend und teils automatisiert durchgeführt. Durch die Verknüpfung in einem iterativen, gesamtheitlichen Prozess wird dadurch eine Evolution erreicht, die in dieser Arbeit unterstützt werden soll.

Zwei für die Arbeit besonders relevante Ansätze sind dabei *Regressionstests* und das *Roundtrip-Engineering*. Regressionstests [AWR91] beschreiben die zyklische Ausführung von Testfällen. Sie sind eine Möglichkeit zu testen, ob das System sich in nicht veränderten Bereichen immer noch korrekt verhält und die Änderung kein fehlerhaftes Verhalten eingeführt hat [MSB11].

Mit Roundtrip-Engineering wird allgemein die Verknüpfung von verschiedenen Softwareartefakten, wie z.B. dem Quellcode oder Modellen verstanden [Gen12]. Dabei kann zwischen drei Phasen unterschiedenen werden: In der ersten Phase des *Reverse Engineering* wird die Verknüpfung des Quellcodes zu einem entsprechenden Modell hergestellt. In der zweiten Phase wird das Modell entsprechend der gewünschten Änderung angepasst. In der abschließenden dritten Phase wird aus dem Modell erneut Quellcode produziert. Dabei wird ein **Prozess** etabliert, indem inkrementell Änderungen zwischen den beiden voneinander unabhängigen Artefakten synchronisiert werden.

Auch die vorliegende Arbeit versucht mit der Koevolution und Fortführung von Modellen einen zyklischen Prozess für die Evolution zu etablieren. Der Prozess ist dabei in Teilen vergleichbar mit der zyklischen Überprüfung auf Fehler bei Regressionstests sowie mit dem Prozess des Roundtrip-Engineerings. Allerdings wird dieser zyklische Prozess nicht auf Basis von Testfällen oder dem Quellcode etabliert, sondern durch Evolutionsschritte auf Basis von koevolvierenden Modellen erreicht.

2.2.3 Modellbasierte und -lernende Ansätze

Bei *modellbasierten* Ansätzen geht es darum Modelle zu einem zentralen oder sogar beherrschenden Aspekt der Spezifikation, der Entwicklung, der Verifikation und Validierung oder dem Betrieb von Systemen zu erheben [E+07]. In diesem Rahmen gibt es verschiedene Forschungsbereiche und Methoden, die sich hinsichtlich des Zusammenspiels zwischen dem Softwaresystem und dem Modell, der Nutzung von Modellierungssprachen und -werkzeugen oder der Form der Formalisierung unterscheiden.

Generell dienen Modelle dazu, eine höhere Automatisierung oder Optimierung des Systems oder seiner Entwicklung bzw. Evolution zu ermöglichen [Har15]. Denn dadurch kann sich der Entwickler mehr auf wertschöpfende Aufgaben konzentrieren und muss sich weniger mit technischen Details beschäftigen [Har15]. Dabei kann allgemein zwischen *modellgetriebenen* und *modellbasierten* Verfahren unterschieden werden. Modellgetriebene Verfahren verfolgen ein hohes Maß an Automatisierung, Genauigkeit und Vollständigkeit, um Softwarecode oder andere Laufzeitartefakte direkt aus Modellen zu generieren. Bei modellbasierten Verfahren wird diese enge Kopplung nicht angestrebt, sondern Modelle werden hauptsächlich zum Verständnis oder der Analyse des Systems verwendet.

Die verwendeten Modelle können verschiedene Sprachen und Notationen verwenden, wie z.B. Zustandsautomaten, Markov-Ketten oder Petri-Netz-Netze. Zusätzlich können auch sogenannte domänenspezifische Sprachen verwendet werden. Diese erlauben es, für ein spezifisches Problemfeld eine formalisierte Sprache zu verwenden, in der domänenspezifische Elemente in Softwareartefakten direkt abgebildet werden [Keh15].

Im Rahmen der Modellbildung spielt in dieser Arbeit das Themengebiet der *modelllernenden* Ansätze eine wichtige Rolle. Auch wenn modelllernende Ansätze an vielen Stellen eingesetzt werden, sind das Lernen von Prozess- und Architekturmodellen von Softwaresystemen als wesentlicher Forschungsbereich zu nennen. Dabei sprechen Cook und Wolf [CW98b] als Erste im Bereich der Softwaretechnik vom Lernen von Prozessen in Softwaresystemen. Die sich daraus entwickelte Forschung zur Entdeckung von Prozessen (*process discovery* oder *process mining*) beschreibt eine Form der Datenanalyse, die den inhärenten Prozess eines Systems extrahiert und in einer passenden Modellierungssprache ausdrückt [CW98b, van11].

Für eine vollständige Modellierung eines Systems sind neben dem Lernen des Verhaltens in Prozessen auch die statischen Verbindungen, Rollen oder Zuständigkeiten zwischen einzelnen Elementen dieses Verhaltens relevant. Unter dem Begriff der Wiederentdeckung von Architekturen (*architecture recovery*) werden Verfahren subsumiert, die Architekturen zumeist aus Quellcode, aber auch aus anderen Informationsquellen, wie dynamischen Ausführungen, historischen Daten oder Versionierungssystemen extrahieren [DP09]. Die verwendeten Methoden entstammen dabei z.B. dem Reverse-Engineering, der Quellcodeanalyse oder Testverfahren [Bin07].

Die vorliegende Arbeit bildet Evolutionswissen in spezifischen Modellen ab und verwaltet diese in einem halbautomatisierten Prozess. Dadurch soll der Nutzer bei seinen wertschöpfenden Aufgaben unterstützt werden, indem er eine Verhaltensbeschreibung und die (nicht)funktionalen Eigenschaften des unterliegenden Systems während der Evolution vom System selbst bereitgestellt bekommt. Dazu werden sowohl modelllernende Verfahren für domänenspezifische Verhaltensmodelle eingesetzt als auch eine Form der statischen Rekonstruktion von Softwarearchitekturen verwendet, um das Verhalten mit Kontextinformationen zu verknüpfen. Die Arbeit fokussiert dabei nach der Kategorisierung von Shi et al. [SWYS11] insbesondere Ereignismodelle für cyber-physische Systeme. Ein Ereignis ist in einem cyber-physischen System dabei als ein softwaretechnisches Abbild eines Vorkommnisses in der physischen Welt zu verstehen [TVG09].

2.2.4 Analyse und Management von Änderungen

Auswirkungen von Änderungen zu bestimmen, kann zwei unterschiedliche Ziele haben: Zum einen können vor der Realisierung einer Änderungen Auswirkungen abgeschätzt werden und zum anderen nach der Realisierung die dadurch entstandenen Auswirkungen analysiert werden [RT01].

Auswirkungen werden typischerweise vor der Realisierung mit Hilfe einer Auswirkungsanalyse (*impact analysis*) abgeschätzt [Arn96]. Ziel dabei ist, Änderungen besser zu verstehen und ihre Kosten besser abzuschätzen [LR+12]. Eine Analyse nach der Realisierung wird als Post-Mortem-Untersuchungen (*post-mortem analysis*) bezeichnet und kann auch zur Laufzeit durchgeführt werden. Dabei können Erfahrungen von Änderungen gesammelt werden, damit diese für folgende Aktivitäten weiter genutzt werden können [SDHM03]. In beiden Fällen können sowohl die funktionalen Eigenschaften als auch die nichtfunktionalen Eigenschaften tangiert werden. Eine Änderung wirkt sich dabei nicht auf jede Eigenschaft gleich aus: So kann beispielsweise die Performanz eines Systems erhöht, aber gleichzeitig die Zuverlässigkeit verringert werden. An dieser Stelle entsteht oft ein Spannungsfeld zwischen zwei gegenläufigen Eigenschaften des Systems. Auswirkungen können dabei auf das untersuchte System oder Artefakt begrenzt sein oder sich auch über verschiedene Systeme oder Artefakte erstrecken.

Neben der Analyse von Änderungen, müssen diese auch verwaltet werden. Im Rahmen der Evolution ermöglicht das Softwarekonfigurationsmanagement, große und komplexe Systeme bezüglich Versionen und Varianten zu unterstützen [CW98a]. Es erlaubt, entstehende Versionen und Varianten des Systems oder einzelner Artefakte zu verwalten und diese kooperativ zu verändern und anzupassen. In diesem Zusammenhang werden in der Softwareentwicklung zumeist Versionsverwaltungssysteme eingesetzt. Diese ermöglichen eine Evolution in Versionen und Varianten von Artefakten konsistent zu halten und die parallele Bearbeitung in Teams zu organisieren [CW98a].

Die im Softwarekonfigurationsmanagement eingesetzten Methoden basieren dabei auf der so genannten Differenzbildung (*differencing*) und der Zusammenführung (*merging*) [Keh15]. Diese Methoden ermöglichen zumeist textuelle Differenzen und ermöglichen eine Zusammenführung von zwei Versionen oder Varianten eines Artefaktes, wie z.B. dem Quellcode oder domänenspezifischen Modellen [Keh15].

Diese Arbeit strebt eine Post-Mortem-Untersuchung von gelernten und vorgeschlagenen Modellartefakten an. Weiterhin wird eine methodische Untersuchung von Versionen und Varianten wie im Softwarekonfigurationsmanagement verfolgt. Beides soll in der vorliegenden Arbeit auf Evolutionsartefakte von cyber-physischen Systemen angewendet werden, um auf Basis dieser Methoden Änderungen zu beschreiben und zu propagieren.

2.2.5 Modelldifferenzen mit semantischem Liften

Da in einem Teil der Arbeit **Modelldifferenzen** eine zentrale Rolle spielen, wird dieser Forschungsbereich mit dem unterliegenden Ansatz des semantischen Liftens von Modelldifferenzen genauer erläutert. Dabei kann zwischen zwei Arten von *Differenzen* unterschieden werden: Zum einen, die in dieser Arbeit verwendeten, nicht kommutativen und asymmetrischen Differenzen, die beschreiben wie aus einem Ursprungsmodell ein Folgemodell abgeleitet wird [Keh15]. Und zum anderen symmetrische Differenzen, die alle Elemente enthalten, welche nur in einem der zwei Modelle vorhanden sind [CW98a].

Gängige Versionsverwaltungssysteme verwenden in der Regel Methoden für eine Differenzenbildung auf Basis von textuell repräsentierten Artefakten, wie z.B. dem Quellcode [PKH+18]. Im Bereich von Modellartefakten gilt dieses allerdings als nicht ausreichend [BE09, EKS09]. Denn um Evolution verständlicher darzustellen werden Werkzeuge auf Modellebene benötigt [Keh15, HCL+17]. Hier folgt die vorliegende Arbeit dem Ansatz der semantisch gelifteten Modelldifferenzen (siehe [KKT11, KKT13, Keh15, PKK+15]), der nachfolgend erläutert wird. Modelle werden darin auf Basis ihrer konzeptionellen Struktur verglichen. Diese ist in Form eines **abstrakten Syntaxgraphen** gegeben, der einem typisierten Attributgraphen entspricht und den Regeln des Meta-Models folgt.

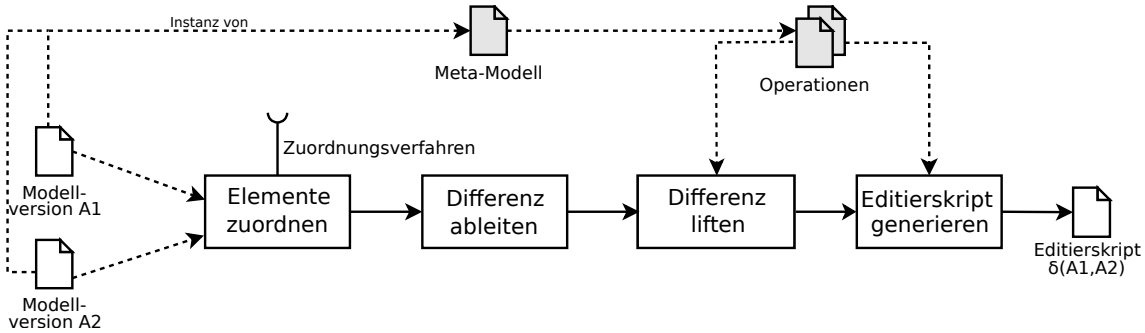


Abbildung 2.1: Vorgehen der Differenzenberechnung (angelehnt an [PKH+18])

Der dabei auf diese Arbeit übertragene operationsbasierte Ansatz ist in Abbildung 2.1 dargestellt. Zunächst müssen hier als *gleich* anzusehende Elemente der beiden verglichenen Modelle identifiziert werden (*Elemente zuordnen*). Die **Zuordnung** ist modellabhängig und kann einzigartige Identifikatoren der Elemente verwenden, sofern diese vorhanden sind und das Element ausreichend charakterisieren [KDPP09]. Wenn dies nicht gegeben ist, können auch ähnlichkeitsbasierte Verfahren eingesetzt werden, die möglichst ähnliche Elemente zuordnen [KKPS12]. Bei der *Ableitung von Differenzen* werden alle nicht zugeordneten Elemente als gelöscht bzw. erzeugt betrachtet. Dies hängt von der Richtung der gebildeten asymmetrischen Differenz ab.

Durch die Anwendung eine zustandsbasierte Differenzenbildung (vergleiche [Lv92, BSW+09]) kann eine Differenz identifiziert werden. Dazu wird die Zuordnungen der Modellelemente verwendet und lokale Attribute dieser Elemente verglichen. Änderungen auf der Ebene des abstrakten Syntaxgraphen werden als **primitive Änderungen** (*low-level changes*) bezeichnet. Dies sind beispielsweise einfache Graphoperationen, die das Hinzufügen und Löschen von Elementen oder Referenzen, wie Assoziationen, Vererbungen oder Kompositionen im Graph vornehmen.

Primitive Änderungen sind für den Nutzer eines Systems, welcher mit dem abstrakten Syntaxgraphen normalerweise nicht vertraut ist, nur schwer zu verstehen [Keh15, HCL+17]. Deshalb folgt diese Arbeit dem Ansatz Differenzen auf die Ebene von **Operationen** (*edit operations*) zu liften (siehe [KKT11, KKOS12, KKT13]. Dabei werden primitive Änderungen in disjunkte Mengen, so genannten semantischen Änderungsmengen (*semantic change sets*), partitioniert. Diese Änderungsmengen müssen genau einer Ausführung einer Operation entsprechen. Wobei Operationen dabei Änderungen auf Nutzerebene beschreiben, die sich an Funktionen von graphischen Modelleditoren orientieren und mit Hilfe von Transformationsregeln auf Basis der einfachen Graphoperationen formuliert werden. Um eine korrekte Anwendung einer Operation zu identifizieren, muss die semantische Änderungsmenge alle in der Transformationsregel der Operation angegebenen primitiven Änderungen enthalten. Ist dies der Fall, wurde mit dieser identifizierten Operationsanwendung die Differenz *semantisch geliftet*.

Durch das semantische Liften wird ein besseres Verständnis von Änderungen ermöglicht, aber noch nicht die Anwendung der gelifteten Operation auf Varianten durchgeführt. Dazu muss zunächst eine partielle Ordnung von Operationen gebildet werden. Entsprechend werden Abhängigkeiten zwischen den Operationen bestimmt und aufgelöst, sowie Parameter der Operation wiederhergestellt (*Editierskript generieren*). Das dadurch erzeugte parametrisierte **Editierskript** beschreibt eine modellbasierte Änderung auf Nutzerebene durch eine partielle Ordnung von Operationen.

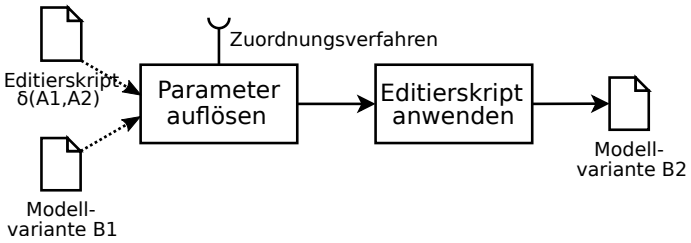


Abbildung 2.2: Vorgehen beim Patchen von Modellen (angelehnt an [PKH+18])

Abbildung 2.2 zeigt, wie ein Editierskript dann auf ein unabhängiges drittes Modell angewendet wird. Die Methode beruht dabei auf den Arbeiten von Kehrer et al. [KKT13, KKK13, Keh15] und wurde auf das Konzept dieser Arbeit zugeschnitten (siehe [PKH+18]). Dazu wird zunächst eine Zuordnung der Parameter des Editierskripts mit dem dritten Modell vorgenommen, indem passende Modellelemente und -attribute für die noch nicht gesetzten Objekt- und Wertparameter identifiziert werden (*Argumente auflösen*). Anschließend kann das Editierskript mit den gesetzten Parametern auf das dritte Modell *angewendet* werden.

Durch diese für die Arbeit auf Evolutionsmodelle von cyber-physischen Systemen angepasste Methode können Differenzen zwischen Modellen auf Operationsebene detektiert, in Form von Editierskripts abgebildet und auf ähnliche Modelle angewendet werden.

2.3 Softwarekonzepte

In der Softwareentwicklung gibt es eine Vielzahl unterschiedlicher Paradigmen, welche jeweils eigene Designphilosophien und Schwerpunkte haben. Die vorliegende Arbeit tangiert verschiedene dieser Paradigmen, die deshalb im Folgenden jeweils kurz vorgestellt werden. Dabei wird jeweils hervorgehoben, welche Aspekte für die im Rahmen dieser Arbeit angestrebte Evolutionsunterstützung wichtig sind.

2.3.1 Verteilte Systeme

Verteilte Systeme beschreiben einen Verbund von Komponenten, in dem lokale und vernetzte Komponenten über Nachrichten kommunizieren, um ihre Aktionen zu koordinieren [CDK05]. Ein verteiltes System erlaubt, unabhängig von dem genauen Standort der Komponenten, Funktionen über ein Netzwerk aufzurufen.

Die Hauptmotivation für verteilte Systeme ist Ressourcen über ein Netzwerk zu teilen [CDK05]. Ressourcen sind dabei als abstrakt anzusehen und umfassen alle Dinge, die über ein Netzwerk von Computersystemen verteilt werden können. Der Forschungsbereich der verteilten Systeme behandelt und löst in diesem Zusammenhang Herausforderungen und Probleme, wie die der Heterogenität, Offenheit, Sicherheit, Skalierbarkeit, Fehlerbehandlung, Nebenläufigkeit und Transparenz dieser als komplex anzusehenden Systeme [CDK05].

Peer-to-Peer Netzwerke beschreiben eine bestimmte Art von verteilten Systemen, die aus unabhängigen, autonomen und gleichberechtigten Knoten (*peers*) mit ähnlichen Fähigkeiten bestehen [CDK05]. Sie fokussieren insbesondere auf den Aspekt der Offenheit, was bedeutet das Knoten dem Netzwerk auf einfache Art und Weise beitreten und dieses auch wieder verlassen können. Beim Austausch von Ressourcen gelten Peer-to-Peer Netzwerke als leicht skalierbar, adaptierbar gegenüber Veränderungen und fair [CDK05]. Durch ihren hohen Verteilungsgrad und die Abstinenz einer zentralen Kontrollinstanz sind diese Netzwerke widerstandsfähig gegenüber Strukturveränderungen und Fehlern einzelner Knoten und können Knoten einen hohen Grad an Autonomie und Anonymität bieten [CDK05].

In einem Peer-to-Peer Netzwerk können verschiedene Technologien genutzt werden, um Daten auszutauschen. Eine Möglichkeit dazu ist eine *Blockchain*, welche eine dezentrale Datenstruktur darstellt, die es ermöglicht, ohne zentrale Instanz eine Konsistenz von Ressourcen zwischen den verschiedenen Knoten eines Peer-to-Peer Netzwerkes herzustellen [MS15]. Dabei sind die wesentlichen Datenstrukturen einer Blockchain *Transaktionen* und *Blöcke* [PKBL18]. Transaktionen bilden die kleinsten Einheiten, die verarbeitet werden und den Zustand der Blockchain verändern [PKBL18]. Transaktionen werden in Blöcken gebündelt, indem diese über Hashes in einer Folge miteinander verknüpft werden. Durch die Verwendung eines *Konsensalgorithmus* wird innerhalb des Peer-to-Peer Netzwerkes ein Knoten bestimmt, der aus den verfügbaren Transaktionen, welche in einem *Mempool* gespeichert werden, jeweils einen neuen Block vorschlägt. Nachdem ein solcher Konsens gefunden ist, wird dieser Block der Blockchain angefügt. Dadurch wird durch die Blockchain eine im Nachhinein unveränderliche Datenstruktur von Transaktionen erstellt.

Für eine Evolutionsunterstützung in cyber-physischen Systemen ist für den Ressourcenaustausch ein verteiltes System unabdingbar. Peer-to-Peer Netzwerke weisen dabei starke Gemeinsamkeiten zu einem (gemeinschaftlichen) Evolutionsprozesses auf, da evolvierende Versionen und Varianten als gleichberechtigte, funktionsähnliche Knoten aufgefasst werden können. Eine Blockchain ist demnach, besser als z.B. verteilte Datenbanken, dafür geeignet, diesen Evolutionsprozess zu speichern und über ein Netzwerk zu teilen.

2.3.2 Komponenten, Dienste und Agenten

Komponentenbasierte Softwareentwicklung kennzeichnet die Komposition von wiederverwendbaren, lose gekoppelten **Komponenten**, die Funktionalitäten mit einer hohen Kohäsion zueinander in einer logischen Einheit zusammenfassen [SBW99]. Eine Komponente ist dabei eine eigenständige Einheit mit klaren Schnittstellen [PB13]. Eigenständigkeit bedeutet, dass sich Komponenten auf die Realisierung und Kapselung von domänenspezifischen Verhalten konzentrieren und nichtfunktionale Aspekte kontextabhängig zur Ausführung konfiguriert werden können [PB13]. Die Ausführung von Komponenten folgt dabei einer Umkehrung des Kontrollflusses, indem sie durch die Infrastruktur kontrolliert und aufgerufen werden [Ors17].

Ähnlich wie Komponenten fordert auch das dienstorientierte Paradigma die Bereitstellung von Funktionalitäten (**Diensten**), wobei es explizit auf verteilte Systeme ausgelegt ist. Durch die Aufteilung von Dienstnutzer, Dienstanbieter und Verzeichnisdienst können so Dienste und ihre Funktionalität gesucht und dynamisch gebunden werden [PB13]. Um höherwertige Funktionalitäten zu erreichen, werden Dienste in der Praxis oft in Prozessen komponiert und in eine logische Abfolge gebracht [HS05]. Diese Dienstkomposition in Prozessen kann nach dem Prinzip der *Orchestrierung* oder *Choreographie* erfolgen: Bei der Orchestrierung wird die Komposition von einer Entität durch das Aufrufen anderer Dienste durchgeführt. Die Choreographie komponiert Prozesse hingegen in einer interaktiven und dezentralisiere Sichtweise. Komponenten und Dienste sind konzeptionell eng miteinander verknüpft, weshalb insbesondere in der Industrie auch weitergehende Konzepte, wie eine Architektur von Dienstkomponenten [MR09] Anwendung finden.

Als ein weiteres Paradigma, betrachtet die Arbeit *Softwareagenten*. Ein Agent beschreibt nach Wooldridge und Jennings [WJ95] eine situierte Softwareentität, die in seiner Umgebung Aktionen durchführen kann, um ihre Ziele zu erreichen. Ein Agent agiert aufgrund seiner lokalen Zustände und Ziele autonom, was bedeutet, dass er volle Kontrolle über seine Ausführung besitzt. Seine Umgebung wird dabei durch Sensoren in Form von Ereignissen wahrgenommen und mit Aktoren wird diese Umgebung beeinflusst. Agenten können unterschiedliche Ausprägungen besitzen, indem sie unterschiedliche interne Verhaltensmuster aufweisen, die durch eine interne Agentenarchitektur festgelegt wird [PB13].

In der Evolutionsunterstützung dieser Arbeit ist eine domänenspezifische, kontextabhängige Kapselung in wiederverwendbaren Komponenten unter Berücksichtigung von nichtfunktionalen Eigenschaften als geeignete Systemarchitektur für das Management von Artefakten zur Laufzeit anzusehen. Dienstorientierte Ansätze werden insbesondere bei der Bildung von Prozessen zum Evolutionsmanagement genutzt, da die avisierte Evolutionsunterstützung lokal zu orchestrieren und systemübergreifenden zu choreografieren ist. Unterschiedliche Verhaltensmuster sind im Rahmen der Evolutionsunterstützung dazu geeignet verschiedene Arten von Artefakten, die den Zustand eines Systems über Sensoren und Aktoren wahrnehmen und beeinflussen, zu beschreiben. Dabei können Agentenarchitekturen genutzt werden, den Aufbau, die Verarbeitung und die Analyse eines Artefakts zu spezifizieren.

2.3.3 Selbstwahrnehmende und ereignisbasierte Systeme

Eine speziellere Klasse von Softwaresystem, die sich in Teilen mit Multiagentensystemen überdeckt, bilden die **selbstwahrnehmenden** (*self-aware*) Systeme. Selbstwahrnehmung bedeutet hier, dass das Systeme sich selbst beobachtet, um sich und seine Komponente stetig mit verändertem Wissen über seine Umgebung zu aktualisieren [SHEA10]. Diese Systeme können Fehler sowie Probleme selbstständig erkennen [DFRG08]. Nach Mowbray und Bron-

stein [MB05] existieren dabei zwei verschiedene Arten der Selbstwahrnehmung: Zum einen ein *selbstreflektiertes Bewusstsein*, welches durch die Analyse der internen Struktur zur Laufzeit erreicht wird. Zum anderen ein *objektives Bewusstsein*, welches durch die Beobachtung des extern sichtbaren Verhaltens entsteht.

Aufbauend auf dem Bewusstsein über sich selbst, existieren eine Vielzahl von Ansätzen für weitergehende Eigenschaften dieser Systeme (vergleiche [BG09]). Beispielhaft seien hier selbstdiagnostizierende, selbstoptimierende, selbstorganisierende und selbstadaptive Systeme genannt. Viele dieser Ansätze, insbesondere der Selbstadaption, folgen dafür der so genannten *MAPE-K*-Schleife (*Monitor-Analyze-Plan-Execute over a shared knowledge*) (siehe [BDG+09]). Diese operationalisiert einen Selbstadaptionsmechanismus durch eine Feedback-Schleife, welche Wissen in den iterativ durchlaufenden Phasen des Beobachtens, der Analyse, der Planung und der Ausführung generiert und verarbeitet [ARS15].

Aufgrund der großen Menge an verfügbaren Informationen ist - neben der Vernetzung - die fortschreitende Automatisierung der Verarbeitung von Daten und Datenströmen entscheidend [MFP06]. Mittels des Konzepts einer systemtechnischen Middleware können dabei Datenströme verwaltet und in den verteilten und zumeist heterogenen Umgebungen eine homogenen Kommunikationsebene etabliert werden. Die Verarbeitung in **Ereignissen** ist hier eine Möglichkeit, heterogene Systeme voneinander zu entkoppeln und eine entsprechende Middleware mit Hilfe von *publish & subscribe Mechanismen* bereitzustellen [MFP06]. Ereignisse beschreiben dabei eine Bedingung oder Zustand in einem beobachteten System oder Komponente.

Die Evolutionsunterstützung dieser Arbeit lässt sich grundsätzlich den Phasen der *MAPE-K*-Schleife zuweisen. Wobei der in dieser Arbeit entwickelte Ansatz nicht die Selbstadaption des Systems als Ziel hat, sondern die Unterstützung der Evolution und das Management von Artefakten für den Nutzer. Entsprechend werden die Phasen der Beobachtung, der Analyse und in einem geringen Teil der Planung tangiert, indem ein objektives Bewusstsein gegenüber sich selbst und den an der Evolution beteiligten Versionen und Varianten verfolgt wird. Entgegen dem traditionellen Vorgehen bei selbstwahrnehmenden Systemen wird dieses Bewusstsein allerdings hier nicht zur Entwurfszeit, sondern, wie bei selbstadaptiven Systemen üblich, zur Laufzeit etabliert [dLGM+13]. Dabei ist in einem informationsgetriebenen Szenario, wie dem der Evolutionsunterstützung, eine ereignisbasierte Kommunikation besser geeignet als anfragebasierte Systeme [FZ97]. Deshalb wird eine ereignisbasierte Kommunikation in dieser Arbeit als unterliegende Systemstruktur angenommen.

2.3.4 Aktive Komponenten

Aktive Komponenten beschreiben ein spezielles Paradigma verteilter Softwareentwicklung. Dieses verknüpft Agenten, Dienste und Komponenten in einem übergreifenden Konzept. Da es im Rahmen dieser Arbeit als zentrales Konzept zur Beschreibung der Architektur genutzt wird, wird es nachfolgend noch etwas detaillierter erläutert. Die Entwicklung von *Aktiven Komponenten* basiert auf zwei zentralen Annahmen, die auch in dieser Arbeit gelten (siehe [PB11]): Zum einen wird angenommen, dass sich Szenarien der realen Welt, dem Agentenparadigma folgend, besser durch aktive und passive Softwareeinheiten abbilden lässt, als es in reinen objekt- oder komponentenorientierten Systemen der Fall ist. Zum anderen sind Systeme einfacher durch Prozesse zu beschreiben, die sich durch die dynamische Auswahl und Bindung von Diensten mit klar abgegrenzten Funktionalitäten zusammensetzen. Diesen Annahmen folgend ist eine *Aktive Komponente* als „eine autonome, verwaltete und eventuell hierarchische Softwareeinheit zu verstehen, die in einer internen Architektur als Komponente definiert ist und imstande ist, Funktionalitäten über Dienste zu nutzen.“(übersetzt nach [PB13])

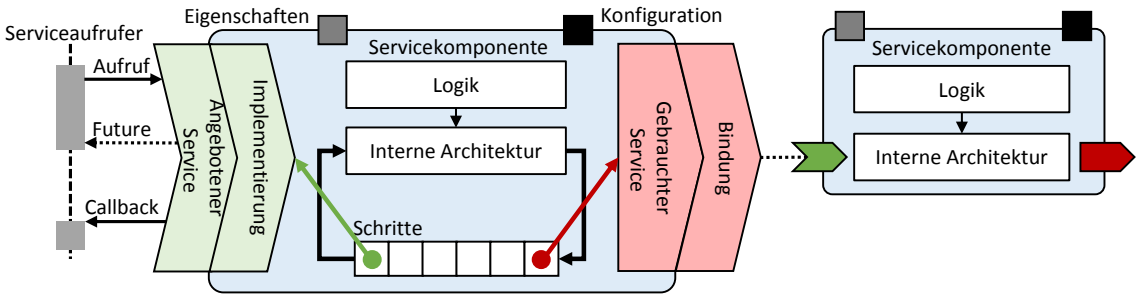


Abbildung 2.3: Aufbau einer Aktiven Komponente (vereinfacht nach [PB13])

Wie Abbildung 2.3 zeigt, ist eine *Aktive Komponente* eine Dienstkomponente, die **angebotene** (*provided*) und **benötigte Dienste** (*required*) spezifiziert. Dienste stellen eine wohldefinierte Funktionalität der Komponente dar und können in beliebiger Anzahl angeboten und genutzt werden. Die Konfiguration der Komponente und ihre (nichtfunktionalen) Eigenschaften können von außerhalb gesetzt und verändert werden.

Die fachliche Logik der Komponente wird dabei - vergleichbar zum Agentenparadigma - auf Grundlage einer internen Architektur beschrieben. Diese beinhaltet ein Ausführungsmodel für den Typ der Komponente und bestimmt, wie die Komponente auf interne und externe Ereignisse reagiert. Mögliche Architekturen gehen von passiven Komponenten bis hin zu einfachen reaktiven oder komplexeren deliberativen Agentenarchitekturen und auch Workflows. *Aktive Komponenten* steuern ihre Ausführung in einem eigenen Ausführungskontext über Schritte, die von einem Interpreter in einer Warteschlange verwaltet und ausgeführt werden. Dabei können sowohl interne Schritte aufgrund einer vorangegangenen Aktion als auch externe Schritte auf Grund eines extern getriggerten Verhaltens eingefügt werden. Das Verhalten des Interpreters und damit die Semantik der Schritte werden durch die interne Architektur bestimmt.

Dienste werden in *Aktiven Komponenten* durch eine entsprechende Dienstschnittstelle spezifiziert. Diese erlaubt einen objektorientierten Zugriff und ermöglicht eine globale Dienstsuche bezüglich des entsprechenden Interfaces [PB13]. *Aktive Komponenten* verarbeiten Dienstaufrufe asynchron unter Verwendung des Aktormodells (siehe [HBS73]). Das Aufrufschema basiert dazu auf Platzhaltern, sogenannten **Futures**, für noch nicht bekannte Ergebnisse [BP11]. Dies ermöglicht eine nichtblockierende Verarbeitung von Ereignissen und erlaubt es den *Aktiven Komponenten* Dienstaufrufe voneinander zu entkoppeln. Eine geschützte **Ausführungsumgebung** entsteht auch beim Aufruf eines benötigten Dienstes, der entsprechend der Dienstsuche an die Komponente zur Laufzeit statisch oder dynamisch gebunden werden kann. Futures können wie in ereignisbasierten Systemen durch ein *publish & subscribe Mechanismus* genutzt werden, wodurch externe Ereignisse regelmäßig wahrgenommen und beobachtet werden können.

Die an der Universität Hamburg entwickelte Systemplattform *Jadex* [PB09] dient als Implementierung und Plattform des Paradigmas von *Aktiven Komponenten*. Jadex umfasst eine verteilte Laufzeitumgebung über Plattformen, die auch die Dienstsuche und -aufrufe übernehmen. Die Jadex-Laufzeitumgebung enthält zudem Hilfsfunktionen, mit denen *Aktive Komponenten* debuggt, bereitgestellt und ausgeführt werden können.

Mithilfe des Paradigmas der *Aktiven Komponenten* können damit Artefakte der Softwareevolution gekapselt und durch fachliche *Aktive Komponenten* über Dienste lokal verwaltet, sowie global über Plattformen ausgetauscht werden.

2.4 Anwendungsgebiete

Nachfolgend werden mit Produktions- und Informationssystemen zwei Anwendungsgebiete für cyber-physische Systeme vorgestellt. Dazu werden entsprechende Funktionalitäten und Terminologien dieser Domänen beschrieben und es werden damit einhergehende Anforderungen an eine Evolutionsunterstützung aufgezeigt. Anschließend wird die Verbindung zu cyber-physischen Systemen aufgezeigt, die Forschungsgegenstand dieser Arbeit sind.

2.4.1 Produktionssysteme

Unter *Produktionssystemen* wird eine Vielzahl verschiedenartiger Systeme zusammengefasst, wobei u.a. zwischen fertigungstechnischen, verfahrenstechnischen und energietechnischen Systemen unterschieden werden kann [Lad18]. Dabei ist ein Produktionssystem als ein mechatronisches System zu verstehen, welches eine funktionale und räumliche Integration von zumeist technischen Komponenten vornimmt, welche Teile aus den Disziplinen der Mechanik, Elektrotechnik und Software vereinen [Ver14]. Das Verhalten ergibt sich aus dem Zusammenspiel dieser Disziplinen, wodurch Produktionssysteme durch eine inhärent hohe Komplexität gekennzeichnet sind [VHDF+14].

Ein Produktionssystem hat rein mechanische Aspekte und Komponente, wie etwa Maschinen oder Fließbänder. Diese sind über elektrische Signale mit einer oder mehreren speicherprogrammierbaren Steuerungen verbunden. Diese Verbindung erlaubt in einem mechatronischen System die Beeinflussung und Kontrolle des Anlagenzustandes und damit der Funktionalität. Dazu werden Wertveränderungen bei den in der vorliegenden Arbeit betrachten Systemen als Ereignisse ausgetauscht. Die Logik dieser Beeinflussung ist zumeist in einer domänenspezifischen Sprache in Softwarecode implementiert. Die Software verarbeitet dabei Ereignisse der Sensoren und reagiert auf diese in der Regel durch die Veränderung von Aktorgrößen. Dies wird in einer speicherprogrammierbaren Steuerung durch zyklisches Verhalten in Realzeit realisiert. Neben dieser untersten Feldebene ist ein Produktionssystem im praktischen Einsatz an weitere informationsverarbeitende Systeme, wie z.B. *SCADA*-, *Prozessleit*- oder *ERP*-Systeme, angebunden.

Die Bewertung von Evolution kann durch nichtfunktionale Eigenschaften vorgenommenen werden [Lad18]. Diese Metriken nichtfunktionaler Eigenschaften eines Produktionssystems können in zeitbezogene Größen, Produktionsvolumen, Topologiegrößen und Fähigkeitsgrößen unterteilt werden (vergleiche [Lad18, LFHL13]).

Evolution ist inhärenter Bestandteil der langlebigen Produktionssysteme, da sie durch neue Anforderung und divergierende Lebensdauern ihrer Komponenten ständigen Änderungen unterliegen [VHFST15, LHFL14a]. Dabei ist es, bedingt durch die hohe Komplexität, schwer Änderungen in den interdisziplinären Komponenten vorherzusagen, denn Änderungen in einer Disziplin ziehen oft Änderungen in anderen Disziplinen nach sich [JFWL11, VHFST15].

Entsprechend werden in der vorliegenden Arbeit Produktionssysteme mit ihrer interdisziplinären Dynamik als Anwendungsgebiet von cyber-physischen Systemen genutzt. Dabei wird insbesondere die Gewinnung, Verwaltung und Weiterentwicklung von Wissensartefakten der Evolution, sowie die Bewertung durch nichtfunktionale Eigenschaften im Rahmen der Evolution betrachtet.

2.4.2 Informationssysteme

Informationssysteme bezeichnen soziotechnische Systeme, welche zusammenarbeitende Komponenten verwendet, um Informationen zu sammeln, zu verarbeiten, zu speichern und zu verteilten [OM05]. Um den Informationsfluss zu gewährleisten, werden Datenbestände zumeist über einen langen Zeitraum verarbeitet [GRG+15]. Das horizontal als auch vertikal integrierte Informationssystem ist dabei durch eine hohe Verflechtung von unterschiedlichen Hard- und Softwarekomponenten gekennzeichnet [HMN15]. Entsprechend sind viele Informationssysteme cyber-physische Systeme und weisen eine hohe Lebensdauer auf [HGH+15, AFG+10].

Ein zusätzliches Element bringt für Informationssysteme der stetige Trend zu verteilten Softwareanwendungen und Cloud-Technologien mit sich [ZZCH10]. Denn dadurch werden zuvor lokal agierende Informationssysteme zu cloud-basierten Informationssystemen. Diese speichern und verarbeiten Informationen geographisch über mehrere Orte verteilt, um durch die Verteilung und Nutzung von entfernten Rechensystemen die Kosten, Verfügbarkeit und Latenz bei großen Nutzerzahlen zu verbessern [RPH+17].

Durch die lange Lebensdauer sind Informationssysteme in der Regel von einer Vielzahl von Änderungen im Laufe ihres Lebenszyklus betroffen [HGH+15]. Diese Änderungen umfassen u.a. die Erweiterungen oder die Änderungen ihrer Anforderungen und die Wartung durch Fehlerbehebung oder Veränderungen der Umgebung bzw. der verwendeten Systemtechnologie. Die Umstellung auf Cloud-Technologien bringt dabei in der Evolution zusätzliche Herausforderung bezüglich der nichtfunktionalen Eigenschaften, wie z.B. der Performanz, der Sicherheit und der Zuverlässigkeit [HGH+15]. In dieser Arbeit wird diesbezüglich insbesondere die Performanz als Metriken für nichtfunktionale Eigenschaften betrachtet.

Cloud-basierte Informationssysteme werden im Rahmen der vorliegenden Arbeit deshalb als Anwendungsgebiet für ein typisches, langlebiges Softwaresystem verwendet, das als cyber-physisches System zu verstehen ist und damit die Anwendbarkeit der vorgeschlagenen Konzepte auf unterschiedliche Anwendungsbereiche zeigen soll.

2.4.3 Cyber-physische Systeme (CPSe)

Der Begriff des **cyber-physischen Systems (CPS)**[2] wurde durch Lee et al. [Lee06] geprägt, um die Interaktion zwischen einerseits der (virtuellen) Software und anderseits der physischen Welt zu betonen. Entsprechend sind CPSe durch „eine Verknüpfung von physischen Objekten und Prozessen mit informationsverarbeitenden virtuellen Objekten über offene, teilweise globale und jederzeit miteinander verbundene Informationsnetze gekennzeichnet“[GB12]. Dabei sind CPSe mehr als ein thematisches Gebiet zu sehen, das Überlappung zu verschiedensten Anwendungsgebieten besitzt [WTO15].

Eine Motivation für derartige Systeme ist es, intelligente Funktionalitäten in traditionell hardwaredominierte Anwendungsgebiete, wie z.B. eingebettete Systeme, Roboter oder Produktionssysteme, zu bringen [DLV12]. Denn durch die Vernetzung lassen sich die eingebundenen Systeme und Komponenten beispielsweise über komplexe, verteilte Infrastrukturen beobachten und beeinflussen. Dabei werden angesichts der hohen Dynamik und Unsicherheit von CPSen u.a. Lern- und Data-Mining-Technologien eingesetzt. Diese dienen dazu nützliche Informationen zu extrahieren und mit Hilfe des Feedbacks des Nutzers neues, unbekanntes Wissen zu gewinnen [WKT11] und auf eine höhere Abstraktionsebene zu bringen [DC14]. Mögliche Modelle von CPSen umfassen dabei sowohl Modelle des physischen Prozesses als auch Modelle der Software, Informationsverarbeitung und des Netzwerks [DLV12].

[2] Im weiteren Verlauf der Arbeit werden Cyber-physischen Systeme als CPS abgekürzt.

Durch die zur Entwicklungszeit nicht vollständig voraussehbaren Interaktionen entsteht während der Evolution bei CPSen ein besonders hoher Bedarf an Flexibilität, lokaler Intelligenz und Adaptionsfähigkeit [BK13]. Dies erfordert zur Evolutionsunterstützung insbesondere die Abbildung des zeitlich-räumlichen Kontextes, kann aber auch den Aufgaben- und Umweltkontext, wie auch den persönlichen oder sozialen Kontext eines Nutzers oder des Systems selbst beinhalten.

CPSe sind in der Regel als offene, soziotechnische Systeme anzusehen, die durch die Nutzung von Informations- und Kommunikationstechnologien auch eine Verbindung zur sozialen Welt herstellt. Dabei sind einzelne Komponenten häufig als Softwaredienste in eine Cloud-Architektur eingebunden [WW18]. Menschen können auf ein CPS einwirken, indem sie Informationen abrufen und eingeben, sowie Abläufe kontrollieren, konfigurieren und steuern.

Um die große Anzahl an verschiedenen Domänen und Bereichen zu kategorisieren, nennen Tongren et al. [TBC+14] einige Unterscheidungsmerkmale, die folgend zusammengefasst dargestellt sind:

- Hardwarenähe: Das Merkmal beschreibt, ob das CPS eng mit einem physischen Prozess gekoppelt ist oder durch die Informationstechnologie dominiert wird.
- Offenheit: In diesem Merkmal wird unterschieden, ob das CPS für eine Vielzahl von potentiell unbekannten Nutzern geöffnet ist oder die Nutzer auf eine spezielle Gruppe beschränkt sind.
- Domänengebunden: Mit dem Merkmal wird angegeben, ob ein CPS in einer speziellen Domäne verordnet ist oder domänenübergreifende Funktionalitäten besitzt.
- Automatisierungsgrad: Durch dieses Merkmal wird bestimmt, ob das CPS einen voll- bzw. teilautomatisierten Ablauf verfolgt oder an eine manuelle Benutzung gebunden ist.
- Verteilung: Das Merkmal gibt an, ob das CPS hauptsächlich zentral gesteuert wird oder ein dezentraler Steuerungsansatz vorliegt.
- Adaptivität: Mit dem Merkmal wird beschrieben inwiefern sich das CPS bei sich ändernden Umgebungen selbständig adaptiert.
- Integration: Das Merkmal unterscheidet, was für einen Grad von horizontaler bzw. vertikaler Integration von Komponenten im CPS vorherrscht.

Entsprechend dieser Kategorien kann ein CPS klassifiziert werden, wobei keine scharfe Trennung der Kategorien erfolgen muss und die Domäne einen maßgeblichen Einfluss auf die jeweilige Kategorisierung hat [WW18].

Entsprechend können die vorgestellten Informationssysteme, ohne weitere Berücksichtigung der konkreten Ausprägung des Systems, als ein CPS mit traditionell softwarekontrollierten Komponenten gelten (vergleiche [Wol09]). Bei Anwendung von Cloud-Technologien lässt sich an dieser Stelle von **cloud-basierten cyber-physischen Systemen (CBCPS)** sprechen. Produktionssystemen haben einen Schwerpunkt auf physischen Maschinen und sind als eine spezielle Untermenge von industriellen CPSen zu sehen [BRTD16]. In diesem Zusammenhang werden Produktionssysteme mit dem Begriff der **cyber-physischen Produktionssysteme (CPPS)** charakterisiert.

Dabei sei an dieser Stelle betont, dass diese ausgewählten Anwendungsgebiete für CPSe nur als Beispiele der entwickelten Evolutionsunterstützung in dieser Arbeit verwendet werden. Entsprechend zielt die vorliegende Arbeit auf keine spezifischen Fragestellungen von CPPSen, wie z.B. die Echtzeitunterstützung oder die Entwicklung von interdisziplinären Modellen, und ebenso von CBCPS, wie z.B. die Bestimmung der Arbeitslast von Anfragen oder die Migration, Replikation oder Bereitstellung von Anwendungen, ab.

2.5 Zusammenfassung

Der Schwerpunkt der Evolution liegt auf der kontinuierlichen Weiterentwicklung des Systems, durch die eine Alterung und negative Auswirkungen auf (messbare) Eigenschaften des Systems begrenzt werden soll. Die Evolution kann durch iterativ verwendete Artefakte und Modelle unterstützt werden, die Änderungen von Versionen und Varianten in über verschiedene Aspekte beschreiben.

Wesentliche Methodiken, die unterschiedliche Typen und Aktivitäten der Evolution unterstützen, sind die iterative Softwareentwicklung, die über wiederkehrende Prozesse die Evolution abbildet, und modellbasierte und modelllernenden Ansätze, die verändertes Verhalten in Modellen darstellen können. Neben dieser Abbildung können Änderungen durch Verfahren, wie der Auswirkungs- und Post-Mortem-Untersuchung, auf Ihren Einfluss, insbesondere von nichtfunktionalen Eigenschaften, analysiert werden.

Durch das Softwarekonfigurationsmanagement stehen Methoden und Werkzeuge zur Verfügung, die Unterschiede von Modellen durch Differenzen beschreiben können. Mit dem semantischen Liften wurde eine Methode vorgestellt, welche Änderungen an einem Modell durch Operationen der konzeptionellen Struktur beschreibt.

Für eine horizontale Verknüpfung in verteilten Systemen stehen Peer-to-Peer-Netzwerke zur Verfügung, die einen Austausch von verteilten Datenbeständen, beispielsweise durch eine Blockchain, erlauben. Komponenten, Dienste und Agenten können in unterschiedlicher Art und Weise zur Unterstützung der Evolution beitragen. Deshalb wurde mit *Aktiven Komponenten* ein Konzept vorgestellt, das fachliche Funktionalitäten in konfigurierbaren, autonom handelnden Komponenten kapselt und über Dienste zugreifbar macht. Für die Evolution können diese vorgestellten Softwarekonzepte genutzt werden, um in ereignisbasierte CPSe durch eine externe Beobachtung ein objektives Bewusstsein über sich selbst und damit auch ihre Evolution zu schaffen.

Cyber-physische Systeme sind ein Bereich der Forschung der durch unterschiedliche Arten von Systemen gekennzeichnet ist. Das wichtigste Merkmal ist eine starke Kopplung von Hardware und Software, dass eine hohe Interdisziplinarität und Komplexität zur Folge hat. Die zwei ausgewählten Anwendungsgebiete - softwareintensiven cloud-basierten Informationssysteme einerseits und hardwarenahe Produktionssysteme andererseits – stellen zwei mögliche Ausprägungen von cyber-physischen Systemen mit jeweils unterschiedlichen Voraussetzungen dar.

3 Modellbasierte Evolutionsunterstützung

Im folgenden Kapitel wird die methodische Fragestellung der Arbeit näher betrachtet.[1] Zunächst werden dafür in Abschnitt 3.1 Hypothesen entlang der Verknüpfungsdimension der Fragestellung aufgestellt, die dann durch das Kapitel im Einzelnen überprüft werden.

Dazu wird in Abschnitt 3.2 das Evolutionsverständnis der Arbeit dargelegt und das Konzept einer schrittbasierten Evolutionsunterstützung über Artefakte eingeführt. Der folgende Abschnitt 3.3 untersucht, welche Wissensartefakte während der Evolution genutzt werden können. Diese Informationen werden in Abschnitt 3.4 verwendet, um einen Prozess zu schaffen, der eine Koevolution von Modellartefakten vornimmt. Dem schrittbasierten Konzept folgend werden in Abschnitt 3.5 Evolutionsschritte aus den Modellartefakten gewonnen, die in Abschnitt 3.6 in einer Vergleichsmethodik genutzt werden, um eine Unterstützung des Evolutionsprozesses von CPSen zu erreichen.

3.1 Hypothesen der methodischen Untersuchung

Unter der identifizierten Problemstellung werden auf Basis der methodischen Forschungsfrage und der zeitlichen und räumlichen Verknüpfungsdimensionen zunächst Hypothesen zur methodischen Lösung formuliert.

Bezüglich der identifizierten Problemstellungen bleibt festzuhalten, dass die Verwendung von Modellen in der Softwareentwicklung als vorherrschende Methodik anzusehen ist, da dadurch die Komplexität eines CPSs durch Abstraktion beherrscht und abgebildet werden kann [BFH+10]. Dabei kann festgehalten werden, dass Modelle für die Konzeption, Implementierung, Prüfung, Optimierung und Diagnose von CPSen geeignet sind [VHDF+14]. In der industriellen Praxis gibt es allerdings aufgrund einer undokumentierte Evolution in der Regel keine aktuellen formalisierten Modelle, die das Verhalten von CPSen ausreichend beschreiben [SS13, Lad18]. Deshalb sollten modellbasierte Methoden auch bei der Erhaltung und Verfügungsstellung von Evolutionswissen und dessen Nutzung eingesetzt werden. Zur Beantwortung der Forschungsfrage in dieser Arbeit wird dabei methodisch nicht, wie oft üblich, auf das Verhalten des CPSs nach der Evolution fokussiert, sondern es wird diese Beschreibung nur genutzt, um die Änderung des Verhaltens als zentrales Element der Evolution zu beschreiben. Daraus ergibt sich die erste Hypothese dieses Kapitels, die in Abschnitt 3.2 methodisch unterlegt wird:

- Eine modellbasierte Beschreibung der Evolution in Evolutionsschritten kann alle unterschiedlichen Aspekte von Änderungen umfassen und stellt damit eine nachvollziehbare Form der Evolutionsbeschreibung dar.

Als wesentliche Problemstellung wurde bereits die Koevolution zwischen System und Artefakten, wie Modellen, identifiziert. Um aktuelle Artefakte auch bei einer unstrukturierten Evolution zu gewährleisten, sollten diese mit Laufzeitinformationen generiert und verknüpft werden. Dies wird als Methodik beispielsweise in der Fehlermodellierung in Architekturmodellen durch probabilistische Fehlerinformation bereits eingesetzt, da dadurch der Einfluss auf die

[1] Eine ergänzende Durchführung der vorgestellten Methoden und Verfahren wird im Rahmen der Evaluation in Machbarkeitsstudien gezeigt (Kapitel 6).

Verfügbarkeit eines Systems analysiert oder die Konsistenz der Modelle besser gewährleistet werden kann (vergleiche [KLM03, BKBR12, MJB+17]). Dies gilt auch für evolvierende CPSe, in denen ein unzureichender Informationsfluss zwischen Entwicklungs- und Betriebsphase innerhalb des Lebenszyklus vorherrscht [LNR11]. Deshalb wurde eine Kombination von modellbasierter Entwicklung und Laufzeitbeobachtungen als wichtige Zielstellung der Evolution identifiziert [LLVH13, HLL+14]. Dazu bedarf es, wie Goltz et al. [GRG+15] festhalten, einer „kohärenten Gesamtmethode, um die einzelnen bereits vorhandenen Methoden der Softwaretechnik miteinander zu verknüpfen“und für CPSe nutzbar zu machen. Dies umfasst insbesondere die Komposition der zur Verfügung stehenden Wissensartefakte. Entsprechend ergibt sich daraus die zweite Hypothese des Kapitels, die in Abschnitt 3.3 und Abschnitt 3.4 methodisch unterlegt wird:

- Eine Koevolution von Modellartefakten kann durch eine Komposition von Wissensartefakten und mit der Verwendung von Methoden der modellbasierten Softwareentwicklung gewährleistet werden.

Durch das Netzwerk eines CPSs ergibt sich die Möglichkeit, Evolution nicht nur durch Verhaltensänderungen zu beschreiben, sondern diese auch auf ähnliche CPSs anzuwenden [LBK15]. Ein geeignetes Ziel ist dabei, wie es Feldmann et al. [FFVH+12] für einen an dieser Stelle vergleichbaren Bereich der Modularisierung formulieren, die Wiederverwendung im Sinne einer Bibliothek von existieren Lösungen. Dabei sollte die Wiederverwendung von vergangenen, ähnlichen Lösungen [VHFF+15] sowie eine geeignete Wahl und Granularität der Lösungen angestrebt werden [MJG11], damit Evolution nicht nur auf der Erfahrung des Nutzers eines einzelnen CPSs beruht [BS09]. Die Arbeit untersucht deshalb, ob ein vergleichendes Verfahren zwischen vormals isolierten Systemen und Komponenten etabliert werden kann, um die Relationen zwischen Gründen, Aktionen und Zielen von Änderungen über Systemgrenzen hinaus zu propagieren. In Abschnitt 3.4 und Abschnitt 3.5 wird deshalb die folgende dritte Hypothese dieses Kapitels methodisch unterlegt:

- Modellbasierte Evolutionsschritte können durch die Darstellung von Gründen, Reaktionen und Auswirkungen einer Evolution auch systemübergreifend den Nutzer bei der angestrebten Evolution unterstützen, sofern die Bedingungen der Evolution vergleichbar sind. Diese Unterstützung kann systemübergreifend und systematisch in einem Evolutionsprozess dokumentiert werden.

3.2 Evolution mit Evolutionsartefakten

Um die erste Hypothese zur geeigneten Beschreibungsform zu prüfen, wird nachfolgend zunächst das zugrunde liegende Konzept einer schrittbasierten Evolution erläutert. Dazu muss zunächst geklärt werden, was in dieser Arbeit unter Evolution verstanden wird. Darauf aufbauend wird eine Beschreibung eines Evolutionsprozesses definiert, die zentralen Artefakte von Evolutionsschritten erläutert und eine formale Beschreibung dazu vorgestellt.

3.2.1 Evolutionsverständnis

Wie im Grundlagenkapitel dieser Arbeit bereits aufgezeigt, existieren unterschiedliche Definitionen des Begriffs der Evolution. Diese Arbeit verwendet die in Tabelle 3.1 vorgestellten Begrifflichkeiten.

Aktivitäten	Änderungen	Erweiterung / Reduktion
		Verbesserung / Verschlechterung
	Wartung	Erhaltung
		Korrektur
		Sanierung

Tabelle 3.1: Begrifflichkeiten der Evolution

Alle durchgeführten Vorgänge der Evolution werden als *Aktivitäten* verstanden. Diese werden bezüglich des Verhaltens eines CPSs definiert, was die wahrnehmbaren Aktionen und ihre Verknüpfungen umfasst. [2] *Änderungen* beschreibt dabei Aktivitäten, welche das Verhalten eines CPSs modifizieren, während die *Wartung* dieses Verhalten (typischerweise) lediglich erhält. Bei Änderungen wird zwischen der *Erweiterung* bzw. *Reduktion* unterschiedenen, in denen der Umfang der Systemfunktionalitäten verändert wird und der *Verbesserung* bzw. *Verschlechterung*, welche Anpassungen beschreiben, die den Erfüllungsgrad einer vorhandenen Systemfunktionalität verändern.

Bezüglich der Wartung wird zwischen *Erhaltung*, *Korrektur* und *Sanierung* unterschieden. Die Erhaltung sorgt dafür, dass ein Verhalten bestehen bleibt. Dies ist insbesondere bei der Betrachtung von Hardware relevant, in denen es zu natürlicher Abnutzung kommt. Die Korrektur beschreibt das Beheben eines Fehlers, ohne dabei die Menge, den Umfang oder den Erfüllungsgrad der Funktionalität zu beeinflussen. Eine Sanierung ist eine Korrektur, die einen relevante Teile der Konstruktion des CPSs verändert. Dabei sind Wartung und Änderungen nicht disjunkt, wodurch z.B. aufgrund einer Wartung auch eine Anpassung des Verhaltens auftreten kann, was dann als eine gleichzeitige Änderung betrachtet wird. Die initiale Entwicklung eines CPSs ist nicht Teil der Evolution, sondern die Evolution folgt der initialen Entwicklung, wenn das System damit beginnt, seinen Hauptzweck, also die Ausführung seiner Funktionalität, zu erfüllen.

Die Evolution wird in dieser Arbeit als ein gesamtheitlicher Prozess aller beteiligten Disziplinen und Artefakte betrachtet. Im Falle von CPSen bedeutet dies, dass Evolution sowohl Software, Hardware als auch ihre Umgebung umfasst. Dies schließt dabei auch, wie von einigen

[2] In vielen Arbeiten werden diese Begriffe bezüglich der Funktionalität und nicht des Verhaltens, wie für diese, auf Verhaltensbeobachtungen basierende, Arbeit, definiert.

Autoren propagiert, das Wissen über die Evolution mit ein (vergleiche [Jaz05, TN14]). Entsprechend sind so auch Entwicklungs- und Laufzeitartefakte Teil der Evolution eines CPSs, welche jedoch parallel zum System einer Koevolution bedürfen. Evolution wird dem entsprechend in dieser Arbeit, wie folgt definiert:

> *Die Evolution beschreibt den Änderungsprozess im gesamtheitlichen Verhalten aller am System beteiligten Disziplinen und Artefakte, der mit der operativen Phase des Lebenszyklus beginnt und mit seiner Stilllegung endet.*

Die gegebene Definition definiert Evolution auf Grundlage von Änderungsaktivitäten und exkludiert damit Wartungsaktivitäten, welche unter der Definition der Arbeit zu keinen am Verhalten des CPSs sichtbaren Änderungen führen.

3.2.2 Verständnis eines cyber-physischen Systems

Wie bereits im Grundlagenkapitel aufgezeigt, ist der Begriff des CPSs dehnbar, da er in der Literatur nur unscharf definiert wird. Entlang der Eigenschaften von CPSen betrachtet die Arbeit hierbei mit den cyber-physische Produktionssysteme (CPPS) und den cloud-basierte cyber-physische Systeme (CBCPS) zwei Anwendungsgebiete von CPSen. Diese sind für CPSe wie folgt zu kategorisieren: CPPS sind nach der Eigenschaftsdefinition von Torngren et al. [TBC+14] (siehe Abschnitt 2.4.3) durch starke hardwarenähe, geringere Offenheit und einen hohen Automatisierungsgrad gekennzeichnet. Es liegt dabei eine hohe Integration von unterschiedlichen Komponenten, wie einzelnen Maschinen, vor.

CBCPSe sind informationstechnologisch dominiert, wenngleich sie einen hohen Grad an Verteilung, beispielsweise durch Cloud-Komponenten aufweisen. Integration, Automatisierungsgrad und Offenheit sind für diese allgemeinerer Themenklasse stark anwendungsabhängig. Dabei können die Grenzen eines CPSs nicht immer strikt definiert werden, da Komponenten eines CPSs auch jeweils ein eigenständiges CPS bilden können.

Für beide Anwendungsbereiche definiert die Arbeit ein CPS wie folgt:

> *Ein cyber-physisches System beschreibt eine technisch-organisatorische Struktur von vertikal und horizontal verteilten autonomen, aber kooperativen physischen und softwaretechnischen Komponenten, die miteinander und mit dem Menschen kommunizieren.*

3.2.3 Schrittbasierter Evolutionsprozess

Eine zentrale Frage bei der Unterstützung der Evolution von CPSen ist die, wie der Prozess der Evolution eines CPSs, welcher durch seine Versionen und Varianten und dessen Übergänge gegeben ist, beschrieben wird. Die biologische Evolutionsforschung beschreibt den Nutzen einer Betrachtung von Evolution in der „Erklärung, wie (lebende) Dinge sich über einen langen Zeitraum verändern und wie sie zu dem geworden sind was sie sind“[May01]. Im Sinne dieser Definition besteht ein entscheidender Unterschied zwischen der Entwicklung und Evolution. Denn bei der Evolution besteht ein starker Fokus auf der **Änderung**, die weniger den momentanen Zustand beschreibt, sondern vielmehr, wie ein System in diesen Zustand gekommen ist. Diese Erläuterung ist in den Versionen und Varianten eines CPSs häufig nur indirekt gegeben. Dies macht es einem Ingenieur bei der Änderung des Systems oft schwer frühere Designentscheidungen und damit den momentanen Zustand des Systems zu verstehen [JB05]. Entsprechend sollte bei der Beschreibung von Evolution der Schwerpunkt auf der Änderung selbst liegen und nicht, wie bei der Evolution teils suggeriert, auf dem Endresultat der

Anwendung einer Änderung - auch wenn dieses Endresultat zweifelsohne das finale Ziel einer Evolution darstellt. Somit bleibt festzuhalten, dass eine Evolutionsunterstützung nicht alleine auf den Artefakten und seinen Versionen beruhen sollte, sondern vielmehr die Änderungen selbst als Artefakt betrachtet werden sollte.

Entsprechend verfolgt diese Arbeit Änderungen als zentrale Einheit zur Beschreibung von Evolution. Evolutionsprozesse werden dabei nicht durch die verschiedenen Versionen dargestellt, sondern durch die Veränderungen, die zwischen den Versionen besteht. Dadurch soll eine abgrenzbare Beschreibung zwischen Versionen und auch Varianten erreicht werden, die als Vorbedingung zum Verstehen und der Planung von Evolution, insbesondere im Kontext von CPSen, gilt [VHFF+15].

Durch diese Beschreibung soll nicht nur die Evolution dargestellt werden, sondern auch eine Repräsentation des inhärenten Wissens im CPS erreicht werden. Diese Repräsentation verfolgt dabei ein Prinzip, welches vergleichbar zur Deltamodellierung von Softwareproduktlinien ist. Softwareproduktlinien beschrieben eine Menge von Softwaresystemen, die einen gemeinsamen Pool an Funktionalitäten besitzen. Aus diesem werden, jeweils dem Bedarf entsprechend, Funktionalitäten auswählt. Deltamodellierung nimmt dafür eine inkrementelle Komposition von Software in Schritten vor (siehe [BSR04]), indem Softwareproduktlinien durch ein Kernmodell und eine Menge an Deltas beschrieben werden [SBB+10]. Diese Deltas beschreiben jeweils Differenzen, die das Kernmodell erweitern können [SBB+10]. Im Sinne der in dieser Arbeit verfolgten Evolution wird die initiale Entwicklung als erster **Evolutionsschritt** betrachtet, der das initiale System erzeugt. Jede Änderung am System wird als ein weiterer Evolutionsschritt verstanden, der das Verhalten modifiziert. Evolutionsschritte können dabei auch eine parallele Variante des CPSs abbilden, wenn zu Beginn der initialen Entwicklung oder bei Änderungen aus einer vorhandenen Version, die Evolution durch zwei jeweils parallel angewandte Evolutionsschritten voranschreitet. Dies ermöglicht es, dass ein Evolutionsprozess nicht nur sequenziell durch Versionen sondern auch durch parallel existierende Varianten definiert werden kann. Ein **Evolutionsprozess** eines CPSs kann so vollständig durch die sequenzielle und parallele Anwendung von Evolutionsschritten abgebildet werden. Entsprechend wird ein Evolutionsprozess eines CPSs wie folgt definiert:

> *Der Evolutionsprozess eines cyber-physikalischen Systems besteht aus Evolutionsschritten, die sequenzielle Änderungen in Versionen als auch die parallele Evolution in Varianten seiner einzelnen Komponenten abbildet.*

3.2.4 Evolutionsschritte

Ein Evolutionsschritt sollte möglichst alle Eigenschaften einer Änderung abbilden. Dazu wird hier die, bereits im Grundlagenkapitel vorgestellte, Klassifizierung von Buckley et al. [BMZ+05] und Chapin et al. [CHK+01] nach Aspekten einer Änderung verwendet (Abschnitt 2.1). Dem folgend, zeigt Abbildung 3.1 wie diese Aspekte in Evolutionsschritten aufgegriffen werden.

Ein Evolutionsschritt hat immer einen Besitzer, der den Evolutionsschritt durchführt (**wer**). Da Änderung auf das Verhalten des CPSs definiert sind, handelt es sich bei dem Besitzer immer um das CPS oder eine seiner Komponenten, bei der die Änderung aufgetreten ist. Weiterhin ist ein Evolutionsschritt aus zeitlicher Sicht ein Teil des Evolutionsprozesses des gesamten CPSs (**wann**). Der Prozess definiert Vorgänger und Nachfolger eines Evolutionsschrittes. Vorgänger beschreiben dabei die vorangegangene Evolutionsschritte und Nachfolger die nachfolgenden Evolutionsschritte, sofern diese existieren. Wenn mehrere Nachfolger existieren, sind mehrere Varianten aus einer Version des CPSs erwachsen.

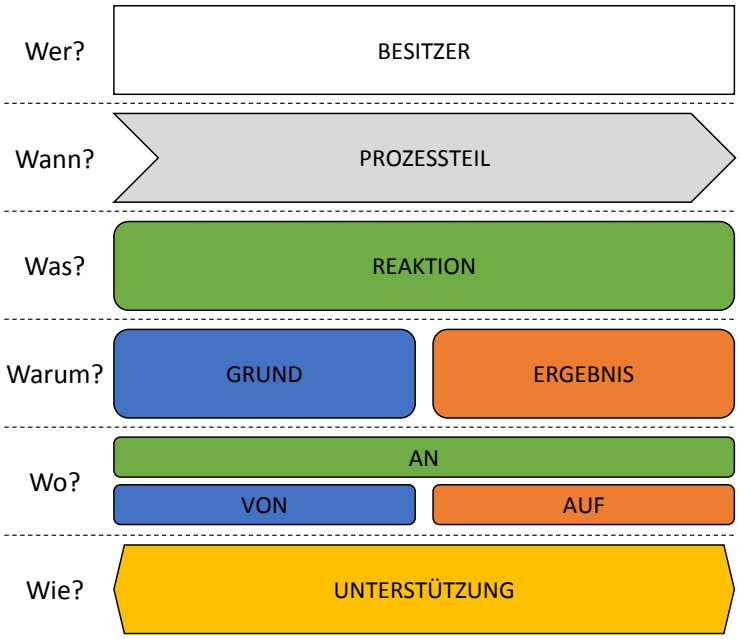


Abbildung 3.1: Konzeptionelle Aspekte eines Evolutionsschrittes

Die eigentliche Beschreibung der Änderung in einem Evolutionsschritt folgt dem CRI-Modell [HPR+18, RPH+17], dass im Rahmen dieser Arbeit entwickelt wurde. Das CRI-Modell (Cause, Reaction, Impact) [3] beschreibt und klassifiziert Änderungen in Grund, Reaktionen und Ergebnis.

Mit dem CRI-Modell können sowohl gewollte als auch ungewollte Änderungen beschrieben werden. Dazu hat das CRI-Modell sechs verschiedene Dimensionen. Diese sind in die drei Phasen *Grund*, *Reaktion* und *Ergebnis* unterteilt, welche jeweils zwei Dimensionen zugeordnet werden (siehe Abbildung 3.2). Die jeweils erste Dimension charakterisiert die Phase und in der zweiten Dimension wird der Ort der Änderung spezifiziert. Die Dimensionen sind unabhängig vom Anwendungsgebiet formuliert und werden durch Kategorien manifestiert. Diese Kategorien werden für die beiden Anwendungsgebiete auf Basis von abstrakten Domänenmodellen formuliert, was als Teil der operativen Fragestellungen in Abschnitt 4.2.2 näher erläutert wird. Die domänenunabhängigen Dimensionen lassen sich jeweils wie folgt beschreiben:

Grund Trigger Mit dem Trigger wird beschrieben, aufgrund was für einer Aktivität die Änderung im System durchgeführt wird. Die Identifizierung des Grunds ist als ein wichtiger Schritt zu sehen, um das unterliegende Motivation der Änderung zu erfassen.

Grund Von In dieser Dimension wird beschrieben worauf sich der Grund der Änderung bezieht. Dies gibt Aufschluss darüber, wofür die betrachtete Änderung gut war. Beispielsweise kann die Änderung aufgrund einer gewollten Verhaltensveränderung des CPSs an sich, eines anderen verbunden Systems oder aufgrund von Eingangs- oder Ausgangsgrößen durchgeführt worden sein.

Reaktion Wirkung Die Reaktion beschreibt, im Rahmen von Änderungen, was sich aufgrund des gegebenen Grunds verändert hat. Dies könnte beispielsweise eine Änderung der Systemlogik sein.

[3] Eine erste Version dieses CRI-Modell entstand im Kontext dieser Arbeit auf Basis einer Masterarbeit von Kim Reichert [Rei17] und wurde in einer Kooperation mit der Adobe Systems Engineering GmbH für Anomalien in Cloud-Systeme entwickelt. In der vorliegenden Arbeit wurde das Modell unter erheblichen Anpassungen auf Evolutionsänderungen übertragen. Details finden sich in [HPR+18, RPH+17]

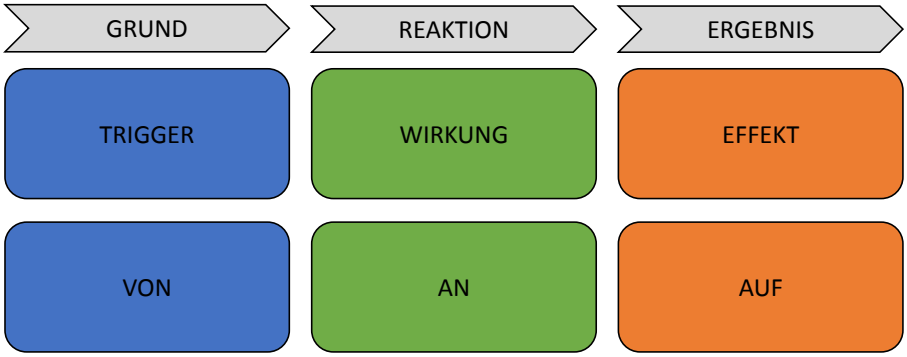


Abbildung 3.2: CRI-Modell für Änderung durch Evolution

Reaktion An Diese Dimension charakterisiert, an welcher Art von Systemkomponenten die Verhaltensänderung auftritt. Dies kann z.B. Aufschluss darüber geben, wie die Wirkung der Reaktion durchgeführt wurde.

Ergebnis Effekt Mit dem Ergebnis wird angegeben, wie sich die Änderung ausgewirkt hat. Dies kann beispielsweise anhand von Metriken der Eigenschaften des CPSs gemessen werden.

Ergebnis Auf Mit dieser Dimension wird beschrieben, auf welchen Teil des CPSs sich das Ergebnis bezieht. Zum Beispiel kann sich das Ergebnis auf nur eine bestimmte Komponente auswirken.

Mit Hilfe des CRI-Modells können weitere Aspekte eines Evolutionsschrittes konzeptionell beschrieben werden (vergleiche Abbildung 3.1). **Was** durch den Evolutionsschritt verändert wird, wird dem CRI-Modell folgend durch eine oder mehrere Reaktionen beschrieben. Durch die Angabe von mehreren Gründen und/oder Ergebnissen wird charakterisiert, **warum** der Evolutionsschritt durchgeführt wurde (Gründe) oder durchgeführt werden sollte (Ergebnisse). Mit Hilfe der drei örtlichen Dimensionen des CRI-Modells wird der Evolutionsschritt bezüglich seiner durchgeführten Änderungen näher in der Komponente lokalisiert (**wo**). **Wie** eine Änderung durchgeführt wird, wird durch die Evolutionsunterstützung definiert, die den Schwerpunkt der nachfolgenden methodischen Betrachtung der Arbeit darstellt.

Um Evolutionsschritte unter der gegeben Sichtweise zum einen gut zu Kategorisierung und zum anderen automatisiert durch eine wissenstragende Komponente zu identifizieren, zu bewerten und zu unterstützen, wird in dieser Arbeit ein halbautomatisierter Ansatz bei der Beschreibung von Schritten verfolgt. Dies soll gewährleisten, dass das als Problem identifizierte Spannungsfeld zwischen dem Produktivitätsgewinn, der durch eine möglichst komplette und qualitativ hochwertige Beschreibung der Evolution entsteht, und der Produktivität, die durch die Erzeugung der Beschreibung verloren geht, angemessen adressiert wird. Entsprechend wird bei der automatisierten Beschreibung ein minimalistischer Ansatz bezüglich der notwendigen Informationen über Evolutionsschritte verwendet, aber gleichzeitig die Möglichkeit geschaffen, zusätzliche manuelle Kontextinformationen mit einzubinden. Diese Kontexte, wie beispielsweise Besitzer, Gründe oder Effekte, sind für den weiteren automatisierten Ansatz der Arbeit nicht unbedingt notwendig, auch wenn diese teilweise automatisch abgeleitet werden können. Dennoch ermöglichen, ggf. manuell durch den Nutzer vervollständigte, Kontexte eine bessere automatisierte Analyse bei der Evolutionsunterstützung und komplettieren die konzeptionelle Betrachtung und Auswertung von Evolutionsschritten. Im Rahmen der operativen Betrachtung behandelt Abschnitt 4.6.3 die Potentiale dieser Kontexte, mit denen beispielsweise passendere Evolutionsschritte durch ihre Relation von Grund, Wirkung und Ergebnis identifiziert werden können.

Zusammenfassend wird ein Evolutionsschritt in dieser Arbeit wie folgt definiert:

> *Ein Evolutionsschritt beschreibt eine modellbasierte, wiederverwendbare Verhaltensreaktion unter einer spezifizierten Kontextbedingung, um eine erwünschte Auswirkung zu erziehen. Durchgeführte Evolutionsschritte können unter Berücksichtigung ihres Besitzers in einem Evolutionsprozess geordnet werden, um so auch zukünftige Änderungen von vergleichbaren Komponenten eines CPSs mit zu unterstützen.*

Ein solcher Evolutionsschritt dient damit als Beschreibung im Sinne der ersten Hypothese zur Methodik. Die methodische Erstellung von Evolutionsschritten aus Artefakten, welche zur endgültigen Bestätigung der Hypothese notwendig ist, wird in Abschnitt 3.5 näher behandelt.

3.2.5 Formalismus des Ansatzes

Die vorliegende Arbeit verwendet für den methodischen Teil die nachfolgend eingeführte semi-formale Beschreibung der Verarbeitung von Ereignissen und ihrer daraus abgeleiteten Evolutionsartefakten. Dabei wird von einer möglichst generischen Betrachtung von ereignisbasierten Systemen ausgegangen.

Das Stream-basierte Verständnis, das zur Modellierung des Unterstützungsprozesses auf Basis von Artefakten genutzt wird, orientiert sich an der *Focus-Theorie* von Broy et al. [BS12]. Diese Theorie wurde ursprünglich für eingebettete Systeme entwickelt, welche sowohl mechanische als auch Softwarekomponenten beinhalten. Deshalb ist diese Theorie an der Schnittstelle zwischen den physischen und virtuellen Komponenten von CPSen besonders geeignet [CIB+14] und wird entsprechend eingesetzt. Die in dieser Arbeit verwendete Notation orientiert sich an der Sichtweise von Schnittstellen und Streams von Komponenten, wendet diese aber auf ereignisbasierte Systeme und ihre Verarbeitung in Artefakten an. Die Modellierung auf Basis von Artefakten folgt dabei dem industriellen Trend, smarte Daten als zentrale Einheit für Innovation in der Digitalisierung zu sehen [OLA+]. Weiterhin wird die Afteraktbeschreibung um eine Verhaltensbeschreibungen ergänzt, die grundsätzlich der Beschreibungsform von UML2 (siehe [RBJ17]) folgt. Dies ermöglicht eine vollständige Beschreibung der Struktur und des Verhaltens von den verschieden methodischen Evolutionsartefakten.

Struktur von Evolutionsartefakten

Evolutionsartefakte beschreiben das vorhandene Wissen über die Systemevolution. Die in dieser Arbeit verwendete Notation beschreibt diese durch ihre **Artefaktstruktur** und ihr **Artefaktverhalten**. Die Artefaktstruktur ist als Beispiel in Abbildung 3.3 visualisiert. Dabei besitzt ein Artefakt disjunkte Eingangsschnittstellen I und Ausgangsstellen O:

$$i_j^A \in I^A \; mit \; 1 \leq j \leq |I^A| \tag{3.1}$$

$$o_j^A \in O^A \; mit \; 1 \leq j \leq |O^A| \tag{3.2}$$

$$I^A \cap O^A = \emptyset \tag{3.3}$$

Der Exponent gibt dabei die Zugehörigkeit zum Artefakt (hier A) an[4] während j einen laufenden Index von I bzw. O darstellt[5].

[4] Sofern die Zuordnung zu Artefakten nicht relevant ist, wird der Exponent von Schnittstellen und weiteren Elementen nachfolgend aufgrund der Lesbarkeit vernachlässigt.

[5] Bei allgemeinen Definitionen, die für beliebige Ein- bzw. Ausgabestellen gelten, wird der Index nicht notiert.

Alle Schnittstellen sind typisiert, was bedeutet, dass sie einer Menge von möglichen **Wertetypen** zugeordnet sind. Diese können Wertetypen, wie ein Ereignis eines Sensors oder Aktors sein, aber auch Werte eines Methodenaufrufes eines Softwaresystems oder Versionen von Modellartefakte. Jede Eingangs- und Ausgangsschnittstelle hat eine mögliche Menge an Typen:

$$T^A_{i^A} \quad \forall i^A \in I^A \tag{3.4}$$

$$T^A_{o^A} \quad \forall o^A \in O^A \tag{3.5}$$

Artefakte werden durch einen **Kanal** $c^A \in C$ aus der Menge aller Kanäle C miteinander verbunden, die jeweils Ein- und Ausgabeschnittstellen verbinden. Diese Schnittstellen müssen zueinander kompatibel sein und definieren damit den Wertetyp des Kanals:

$$T_c = T_i = T_o \; wenn \; i \xrightarrow{c} o \tag{3.6}$$

Der Kanal wird dabei durch seinen Exponenten dem Artefakt seiner Eingangsschnittstelle zugeordnet. Kanäle besitzen als Index zusätzlich einen Namen, der die durch sie transferierten Objekte beschreibt. Dieser entspricht dem Wertetyp der Schnittstelle bzw. des Kanals. Objekte dieses Wertetyps, sogenannte *Wertetypobjekte* werden über die Schnittstelle an andere Artefakte transferiert. Ein Wertetypobjekt besteht dabei immer aus *Daten* und seinem *Kontext*. Daten beschreiben die Nutzinformationen, welche in dieser Arbeit zum einen einfache Informationen, vergleichbar mit Attribut-Werte Paaren, oder zum anderen komplexere Informationen wie Artefakte bzw. Modellinstanzen sein können. Mit dem Kontext werden Eigenschaften und Merkmale dieser Daten beschrieben, die der Interpretation dienen. Beispielsweise einen Namen oder semantische oder syntaktische Informationen. Dies geht einher mit dem halbautomatisierten Vorgehen der Arbeit, indem diese Kontexte zusätzlich durch den Nutzer eines CPSs bereitgestellt werden können.

Artefakt können neben den Wertetypobjekten noch weitere interne logische Datentypen enthalten, die Teil einer Assoziationshierarchie der Wertetypen sein können. Es ist zu beachten, dass an dieser Stelle Werte- und Datentypen als logische Informationseinheiten betrachtet werden, die das Artefakt charakterisieren und in einer logischen *Datenstruktur* beschrieben werden. Diese Datenstruktur beschreibt auf welche Art Daten angeordnet und miteinander verknüpft sind. Die Umsetzung dieser logischen Datenstruktur in softwaretechnische Entitäten, wie u.a. Dienste, Komponenten und Objekte, wird in Kapitel 4 näher betrachtet.

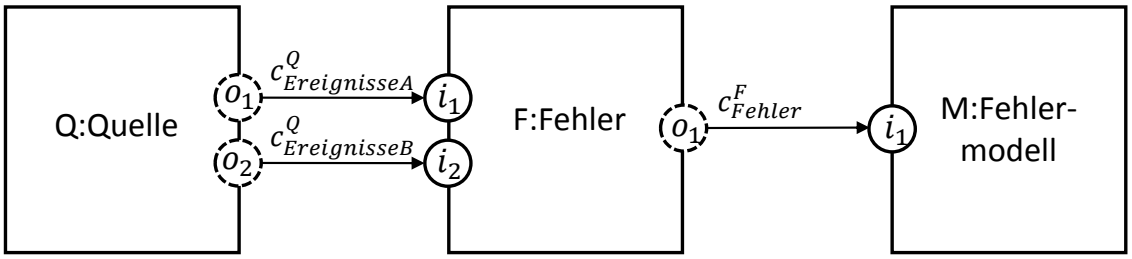


Abbildung 3.3: Notation zur Beschreibung der Artefaktstruktur eines Fehlerartefakts (angelehnt an die *Focus-Notation* nach [MJB+17])

Als Beispiel eines Artefakts beschreibt Abbildung 3.3 das Fehlerartefakt F. Für beide Eingangsschnittstellen i_1 und i_2 des Fehlerartefakts gelten die Typen von Binärsignalen $T_{Binär} \in \{0, 1\} \; \forall i_j \in I$. Die Ausgangsschnittstelle o_1 des Artefakts beschreibt dabei eine Schnittstelle, durch die Fehler an ein Fehlermodell transferiert werden. Diese einzige Ausgangsschnittstelle $o_1 \in O$ hat als Wertetypen $T_{Fehler} \in \{zeitlicher \; Fehler, \; funktionaler \; Fehler\}$.

Die Datenstruktur des Fehlerartefakts wird in der Abbildung 3.4 in Anlehnung an UML2-Klassendiagramme dargestellt. Dabei zeigt die Darstellung den Wertetyp Fehler und seine Ausprägungen als Vererbungshierarchie. Durch die Annotation der Ausgangsschnittstelle wird symbolisiert, dass es sich um einen Wertetyp einer Ausgangsschnittstelle handelt und entsprechend Objekte dieses Typs über den Stream eines Kanals transferiert werden. Im Beispiel kann das transferierte Objekt ein zeitlicher oder funktionaler Fehler sein. Der Wertetyp der Eingangsschnittstelle des Fehlerartefakts ist dabei nicht Teil der dargestellten Datenstruktur, da diese dem Artefakt Quelle (siehe Abbildung 3.3) zugehörig ist. Neben dem Wertetypobjekt wird auch dessen Assoziationshierarchie dargestellt. Abbildung 3.4 zeigt dabei die *Daten* und den *Kontext*, als die zwei Grundelemente der Assoziationshierarchie. Die Daten bestehen aus Fehlerinformationen, beispielsweise in Textform. Dabei gibt die Kardinalität an, wie viele Fehlerinformation möglich sind. Beim Kontext ist im gegebenen Beispiel der Zeitkontext durch genau einen Zeitstempel gegeben.[6]

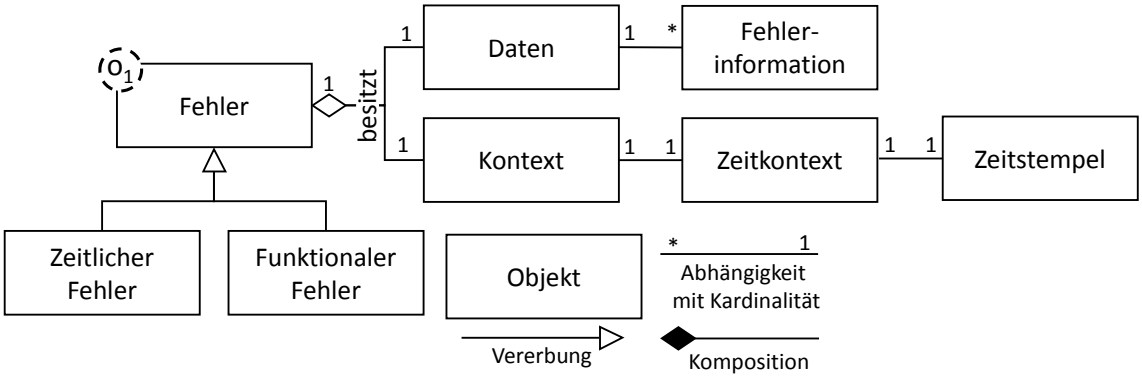


Abbildung 3.4: Notation zur Beschreibung der Datenstruktur eines Artefakts (angelehnt an UML2 Klassendiagramm/Objektdiagramm [RBJ17])

Verhalten von Evolutionsartefakten

Neben der Artefaktstruktur wird auch das **Artefaktverhalten** modelliert. Dazu wird das Konzept von Streams verwendet. Ein Stream ist für jeden Kanal definiert als eine partielle Funktion der Zeit auf Ereignisse ihrer Wertetypen T_c oder das leere Ereignis $\square \notin T_c$:

$$\vec{c} : \mathbb{R}_+ \rightarrow T_c \cup \{\square\} \tag{3.7}$$

Eine Zuordnung von Wert zu Ereignis in diesem Stream wird als Tupel mit der Zeit t und dem Wert eines bestimmten Typs des transferierten Ereignisses des Streams x definiert:

$$(t, x)\ mit\ t \in \mathbb{R}_+ \wedge x\ \in T_c \cup \{\square\} \tag{3.8}$$

Partiell bedeutet in diesem Zusammenhang, dass nicht zu jeder Zeit eine Ausgabe definiert ist. An dieser Stelle ist die Notation vergleichbar zu der oft in ereignisbasierten Systemen verwendeten Notation von Ereignissequenzen. Denn die partielle Funktion kann auch als *Sequenz* oder *Trace* von Ereignissen interpretiert werden. Der Zustand des CPSs ist entsprechend auch in Streams über den Zustand aller Ereignisse bis zu einem bestimmten Zeitpunkt zu verstehen. Dieser ist für den Zeitpunkt t konsistent, wenn alle Ereignisse bis zum Zeitpunkt t empfangen sind. Wie dieser Zustand softwaretechnisch realisiert ist, wird operativ in Abschnitt 4.4.2 diskutiert.

[6] Eine Kardinalität von 1 wird nachfolgend zur vereinfachten Lesbarkeit in einigen Abbildungen vernachlässigt.

Das gesamte Verhalten eines Artefakts wird in einer **Verhaltensfunktion** beschrieben:

$$V^A : I^A \rightarrow O^A \tag{3.9}$$

Abbildung 3.3 notiert unter dieser Definition das Verhalten $V^{Fehlerartefakt} : \overrightarrow{\{i_1^F, i_2^F\}} \rightarrow \overrightarrow{\{o_1^F\}}$. An dieser Stelle sei betont, dass bei mehreren Ausgangsschnittstellen nicht jede Verhaltensfunktion des Artefakts Ereignisse aus T_c in allen Ausgabekanäle erzeugen muss, sondern Definition 3.7 folgend auch ein leeres Ereignis $\square \notin T_c$ beschreiben kann. Intuitiv bedeutetet die Verhaltensfunktion, dass eine Sequenz von Ereignissen gesendet wird, die vom Artefakt über Streams wahrgenommen werden.

Die Realisierung der Verhaltensfunktion ist abhängig von der verwendeten Methode. Im Rahmen dieser Arbeit werden eine Reihe verschiedener Methoden entwickelt oder aus vorhandenen Lösungen adaptiert. Die daraus resultierenden Verhaltensfunktionen werden dabei in Anlehnung an UML2 als Aktivitätsdiagramme dargestellt.

Dies ist in Abbildung 3.5 anhand einer simplen Fehlererkennung für das Fehlerartefakt dargestellt. Der Fehlerprozess hat die **Aktivitäten** *Systemstart*, *Überwachung* und *Meldung des Fehlers*. Diese besitzen zeitliche Verknüpfungen, die den **Kontrollfluss** darstellen und als Pfeile notiert sind. In Abbildung 3.5 werden zunächst der Systemstart und dann die Überprüfungen durchgeführt sowie schließlich etwaige Fehler gemeldet. Der Kontrollfluss kann, wie in UML üblich, Bedingungen haben, die wiedergeben, dass für die Fortführung des Prozesses über diesen Pfeil eine bestimmte Bedingung erfüllt sein muss. Hierbei kann es auch alternative oder parallele Pfade im Kontrollfluss geben. Dabei kann der Prozess auch reflexiv sein, also einen Zyklus beschreiben, wie in Abbildung 3.5 bei keinem erkannten Fehler einer Überprüfung dargestellt.

Die Verbindungen zu den Schnittstellen der Artefaktstruktur sind durch die Notation von Eingangs- und Ausgangsschnittstellen an den Aktivitäten gegebenen. Diese sind vergleichbar zu Ein- und der Ausgabe-Pins des Objektflusses in UML2, aber beschreiben die Schnittstellen hier jeweils mit den dazugehörigen Kanälen und Streams der bereits gezeigten Artefaktstruktur. Über diese Schnittstellen wird das Artefaktverhalten mit der Artefaktstruktur verknüpft. Im Fall des Fehlerprozesses werden die Streams der Eingangsschnittstelle $i_1, i_2 \in I$ bei der Überwachung genutzt und es wird ein Ereignis im Stream der Ausgangsschnittstelle o_1 bei der Meldung des Fehlers generiert indem ein entsprechendes Wertetypobjekt erzeugt wird.

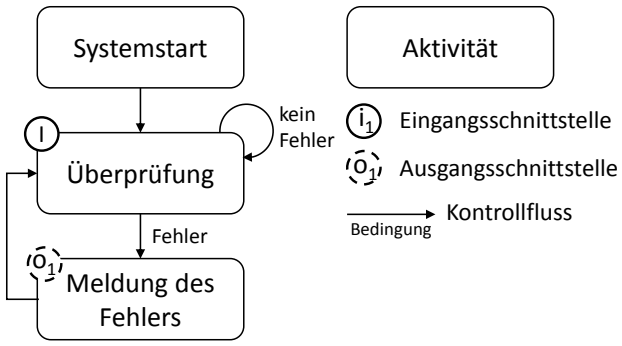


Abbildung 3.5: Notation zur Beschreibung einer Verhaltensfunktion eines Artefakts (angelehnt an UML2 Aktivitätsdiagramm [RBJ17])

Sowohl für die Artefaktstruktur als auch das Artefaktverhalten können unterschiedliche Instanzen der Artefakte vorliegen. Dies ist beispielsweise bei der räumlichen Verknüpfung der Fall, wo zwei ähnliche Modelle von zwei verschiedenen CPSen oder seiner Komponenten existieren. Dies ist in der Artefaktstruktur und auch im Verhalten nicht explizit abgebildet. Dennoch wird die Tatsache mehrerer vorliegender Instanzen von Artefakten in dieser Arbeit berücksichtigt. Dabei werden Artefakte, Objekte, sowie Modell(elemente) eines anderen Artefakts mit einem Tilde-Akzent versehen. Somit lassen sich zwei verschiedene Fehlerartefakte F und $\tilde{F}$ beschreiben, die entsprechend die Schnittstellen i_1 und $\tilde{i_1}$ besitzen.

Dabei werden Artefaktstruktur und -verhalten in semi-formale Diagramme spezifiziert. Sie dienen der Übersicht über die durchgeführten Aktivitäten und können im Sinne von Pseudocode verstanden werden. Entsprechend haben sie keine vollständig ausspezifizierte Ausführungssemantik, sondern sollen vielmehr eine möglichst abstrakte und allgemeingültige Beschreibung der logischen Datenstruktur und des Verhaltens bieten. Die Konzeption eines artefaktbasierten Prozesses und die methodische Realisierung seiner Verhaltensfunktionen unter einer passenden Datenstruktur für CPSe wird im weiteren Verlauf des Kapitels erläutert. Die softwaretechnische Implementierung mit Softwarekomponenten einer wissenstragenden Komponente wird in Kapitel 4 thematisiert.

Zusammenfassung der Evolution mit Evolutionsartefakten

In diesem Abschnitt wurden die Begriffe der Evolution und des cyber-physischen Systems für diese Arbeit definiert. Darauf aufbauend wurde das grundlegende Verständnis eines Evolutionsprozesses erläutert, das auf der sequenziellen Anwendung von Evolutionsschritten fußt. Für diese Evolutionsschritte wurde jeder Aspekt einer Änderung konzeptionell definiert. Diese konzeptionellen Aspekte bestehen insbesondere aus der Reaktion, dem Grund und dem Ergebnis, die durch das CRI-Modell spezifiziert werden. Weiterhin wurde der Besitzer, der Prozess, in dem sich der Evolutionsschritt befindet und die Unterstützung als konzeptioneller Aspekt eines Evolutionsschrittes definiert. Diese konzeptionellen Aspekte von Evolutionsschritte verbleiben allgemein, werden allerdings im Verlauf dieser Arbeit noch weitergehend konkretisiert.

Weitergehend wurde ein semi-formale Beschreibung vorgestellt, durch die Evolutionsartefakte, wie u.a. Evolutionsschritte und ihre konzeptionellen Aspekte, methodisch einheitlich beschrieben werden können. Die Beschreibung umfasst insbesondere die Artefaktstruktur, die Artefakte über Kanäle und unter einer logischen Datenstruktur verbindet und für jedes Artefakt eine Verhaltensfunktion vorgibt. Diese Verhaltensfunktion muss durch konkrete Artefaktverhalten, die auf einer vorhandenen oder neuartigen Methode beruhen können, methodisch unterlegt werden. Die vorgestellte Beschreibung in Artefakten auf Grundlage etablierter Theorien stellt dabei auch für sich bereits einen vom Forschungsthema unabhängigen Beitrag dieser Arbeit dar.

Nachfolgend wird nun die Evolutionsunterstützung in einer aus mehreren Teilen bestehenden Artefaktstruktur erläutert. Diese Teile untersuchen die methodischen Hypothesen, die sich auf die allgemeine, zeitliche und räumliche Verknüpfung der methodischen Forschungsfrage beziehen. Für jedes Artefakt der Artefaktstruktur werden dann nachfolgend jeweils eine oder mehrere Methoden erläutert, die die Verhaltensfunktion durch ein konkretes Artefaktverhalten realisieren.

3.3 Informationsgewinnung in Quell- und Informationsartefakten

Die Evolution folgt unmittelbar der Entwicklung eines CPSs (siehe Abschnitt 3.2.1). Sowohl während dieser Entwicklung als auch während der Evolution ist es damit notwendig, kontinuierlich das System und seine Eigenschaften zu prüfen [RB14]. Die kontinuierliche Betrachtung wird im Folgenden zur Untersuchung der zweiten Hypothese, zur Generierung und Koevolution in Modellartefakten, betrachtet.

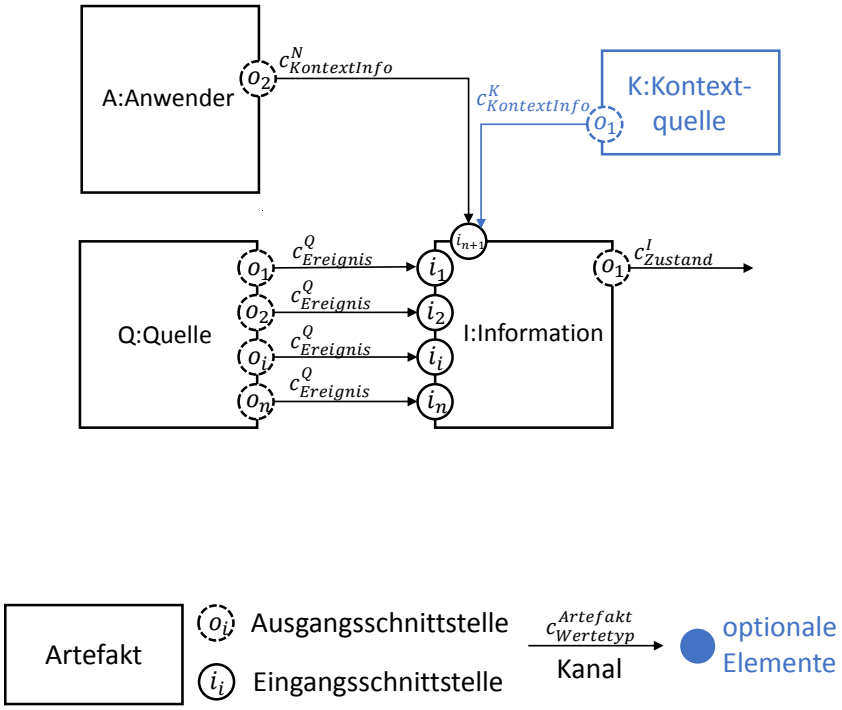


Abbildung 3.6: Artefaktstruktur für die Extraktion für die Informationsgewinnung

Unter der in Abbildung 3.6 gezeigten Artefaktstruktur unterscheidet die vorliegende Arbeit vier Evolutionsartefakte für die Informationsgewinnung. Das erste Artefakt beschreibt die Quellen von Wissen. Ein **Quellartefakt** bezeichnet dabei eine Quelle von Ereignissen, die Informationen über das CPS oder seine Komponenten bereitstellt. Diese können dabei in unterschiedlicher Form vorliegen. Dies sind beispielsweise unstrukturierte oder strukturierte Daten, die zur Steuerung, Kontrolle oder Auswertung des CPSs verwendet werden. Daneben können dies auch Informationen, wie z.B. Kennwerte oder Modelle, sein, die zur Spezifikation, Dokumentation oder Vorhersage genutzt werden. Die Quellartefakte können eine Reihe von Kanälen besitzen, über die Wissen in Form von Ereignissen aus dem Artefakt bezogen werden können. Welche Ereignisse zur Verfügung stehen, hängt dabei von der spezifischen Quelle ab.

Eine weitere Quelle von Wissen ist die menschliche Expertise des Nutzers eines CPSs die das zweite so genannte **Nutzerartefakt** darstellen. Dabei kann ein vom Nutzer zur Verfügung gestelltes Artefakt jedes Wissen enthalten, das durch eine am CPS beteiligte Person gehalten wird. Der Begriff des Nutzers umfasst somit gleichsam die nicht vollständige Liste des Operators, Betreibers, Managers, Entwicklers oder Testers eines betrachteten CPSs. Die Ereignisse aus diesem manuellen Artefakt sollten möglichst gering gehalten werden. Die Aufgaben des Nutzers beschränkt sich deshalb zunächst auf die Bereitstellung von einigen Kontextinformationen über Ereignisse des beobachteten CPSs sowie eine Entscheidung über gewollte oder ungewollte Änderungen.

Sofern nicht das Wissen des Nutzers für Kontexte genutzt werden soll, können auch zusätzliche **Kontextartefakte** in das vorgestellte Konzept integriert werden. Diese optionalen Artefakte können den Nutzer zusätzlich bei Aktivitäten der Evolution entlasten und ermöglichen die Ableitung von einfachen Kontextinformationen über zusätzliche Methodiken. Eine Möglichkeit ist hier beispielsweise die Analyse des Softwarecodes oder die Nutzung von vorhandenen Artefakten der Entwicklung.

Um Evolution methodisch mit Evolutionsartefakten zu unterstützen, müssen sowohl Ereignisse von Quellartefakten als auch die Kontextinformation automatisiert dokumentiert werden. **Informationsartefakte** dienen dafür als Grundlage für das Aufbauen höherwertigen Wissens. Sie sind insbesondere Teil der Informationsintegration in ein CPS und nehmen Quell-, sowie Kontextwissen auf, um Ereignisse im CPS zur Verfügung zu stellen.

Die vier Artefakte zur Informationsgewinnung werden nachfolgenden näher beschrieben und es wird dabei ihre jeweilige Artefaktstruktur und ihr Artefaktverhalten genauer erläutert. Dabei stellt Abschnitt 3.3.1 mit dem operative betriebenen CPS die für die Arbeit verwendete Hauptquelle von Ereignissen vor. In dem folgenden Abschnitt 3.3.2 wird die ergänzende Nutzung von Entwicklungsdokumenten als Quell- oder Kontextartefakten gezeigt. Das Nutzerartefakt und die Informationsartefakte sind dabei Teil der jeweiligen Erläuterungen.

3.3.1 Operative Systeme als Informationsgrundlage für Evolution

Als wichtigste Informationsquelle während der Evolution von CPSen kann die Ausführung des CPSs selbst oder die seiner Komponenten dienen, indem das Verhalten oder die Struktur des CPSs jeweils beobachtet und analysiert wird. Ziel ist dabei immer, dass die Grundlage zur Koevolution von Modellartefakten gelegt wird. Es werden dabei Artefakte für beide Anwendungsgebiet (CPPS und CBCPS) vorgestellt.

Grundsätzlich sind als Quelle jegliche Artefakte denkbar, die Ereignisse in Form eines Streams, also einer stetigen Sequenz von Ereignissen, bereitstellen können. Auf solchen Sequenzen basieren die meisten Ansätzen zur Beobachtung von Systemen und deren Verhalten, weshalb dafür auch eine entsprechend breite Auswahl an Methodiken verfügbar ist. Dabei wird die Auswahl für die Evolutionsunterstützung insofern eingegrenzt, als dass nur Verhaltensänderungen erkannt werden können, die sich aktiv auf die Ereignissequenzen und deren Kontext, wie beispielsweise die zeitliche Performanz, auswirken. Dies kann allerdings bei Änderungen grundsätzlich als gegeben angenommen werden [KIT93], da das (zeitliche) Verhalten von Ereignissen in der Regel alles widerspiegelt, was während einer Ausführung passiert [YPW08].

Das Beobachten eines laufenden CPSs ist dabei ein messbasiertes Verfahren, das zumeist im Betrieb oder während der Evolution, eingesetzt wird [Wal15]. Es schafft die Voraussetzung für den robusten Betrieb und setzt Methoden und Werkzeuge zur effektiven Überwachung des Laufzeitverhaltens voraus [vRH+09], wodurch die Messungen integraler Bestandteil von technischen Prozessen werden [ED07]. In der Praxis konzentrieren sich entsprechende Überwachungslösungen aber oft auf hohe Abstraktionsebenen, wie die Überwachung eines Datenbankservers oder eines Webservers [Roh15]. Dabei sollte, um ein Verständnis und eine entsprechende Dokumentation des Verhaltens zu erreichen, insbesondere bei kontinuierlich eingesetzten Komponenten, das dynamische, steuernde Verhalten überwacht werden [Jon10]. Dies gilt im Besonderen auch für die Evolution, da zu deren Unterstützung ein gleichbleibendes oder besseres Verhalten erreicht werden soll.

In softwarebasierten Systemen können Ereignisse aus einem laufenden System entweder durch die Instrumentalisierung des Softwarecodes gewonnen werden oder das Verhalten wird in der Ausführungsumgebung aufgezeichnet [Kro12]. Erstere Lösung ist nur bedingt in CPSen einsetzbar, da Ihr Verhalten immer aus dem Verhalten von Software und Hardware entsteht. Dennoch kann es für den softwarespezifischen Aspekt von CPSen eingesetzt werden. Die zweite Lösung wird in der Entwicklung oft in speziellen Testumgebungen umgesetzt, ist aber im Evolutionskontext gut geeignet, da in diesem ein laufendes System existiert.

Im Gegensatz zur Konstruktion von Systemen sollte die Überwachung möglichst nicht in die Geschäftslogik eingreifen [vRH+09]. Entsprechend sollte eine Beobachtung einen möglichst geringen zusätzlichen Leistungsaufwand mit sich bringen, obwohl dieser im Vergleich zur unbeobachteten Ausführung durch die Einbindung von zusätzlichem Beobachtungscode häufig unumgänglich ist [Roh15, vRH+09]. Entsprechend ergibt sich die Anforderung, dass eine Beobachtung der Ausführungsumgebung verfolgt werden sollte, die die Implementierung und Wartungsroutinen des Nutzers bei der Steuerung des CPSs nicht oder nur geringfügig beeinflusst. Nachfolgend werden zwei Quellen für CPSe betrachtet, die die genannten Bedingungen erfüllen. Dies ist zum einen die Verarbeitung von Ereignissen auf einem Ereignisbus und zum anderen eine Beobachtung des Kontrollflusses von Dienstaufrufen in Komponenten.

Bussysteme:

Ein Bussystem ist ein gemeinsames Kommunikationsmedium zur Übertragung von Ereignissen zwischen mehreren datenverarbeitenden Komponenten [Int]. Die Beobachtung des steuernden Bussystems ermöglichen dabei, den Kontrollfluss eines CPSs direkt in Ereignissen zu beobachten [GB12]. So beeinflusst die Beobachtung des Systembusses die an der Steuerung beteiligen Komponenten nicht oder nur minimal, weshalb sie insbesondere im eher traditionell geprägten Anwendungsgebiet von CPPSen anwendbar ist. Die resultierenden Quell- und Informationsartefakte eines CPPS-Bussystems werden nachfolgend erläutert.

Für CPPSe werden Feldbussysteme beobachtet, die Bussysteme beschreiben, welche verschiedene Feldgeräte, wie beispielsweise Sensoren und Aktoren, untereinander und mit der speicherprogrammierbaren Steuerung sowie weiteren Automatisierungskomponenten verbinden. Bussystemen basieren auf dem Austausch von Ereignissen, welche immer von allen Teilnehmern gelesen und durch einen Filter zur Verarbeitung eingegrenzt werden. Diese Ereignisse können beispielsweise Signale von Sensoren sein. Für die Arbeit wird angenommen, dass die Ereignisse jeweils durch ein Attribut-Wert-Paar gekennzeichnet sind. Ein Attribut beschreibt ein bestimmtes Signal und der Wert des Signals ist sein elementarer Zustand. Der Wechsel von einem Elementarzustand eines Signals in einen anderen wird durch ein Ereignis, das über den Bussystem beobachtet werden kann, definiert.

Das Bussystem kann als ein Quellartefakt im Sinne der Arbeit verwendet werden. Das dazu realisierte Artefaktverhalten zeigt Abbildung 3.7. Erste Aktivität darin ist die Definition der wahrgenommenen Ereignisse. Dabei wird angegeben, welche Ereignisse auf dem Bussystem im Artefakt abgebildet werden. Im Rahmen von CPPSen ist dies die Definition der Steuerungssignale. Diese werden durch das Quellartefakt wahrgenommen indem es die Ereignisse des Bussystems filtert und die beobachteten Werteobjekte dann über seine Ausgangsschnittstellen transferiert. Neben diesen Ereignissen können Kontextinformationen auch manuell durch den Nutzer definiert werden, was in der Artefaktstruktur durch den Kanal zwischen Kontext- und Quellartefakten modelliert ist (vergleiche Abbildung 3.6). Dabei fokussiert die Arbeit den zeitlich-räumlichen Kontext und den Aufgabenkontext, der für CPPSe als entscheidend identifiziert wurde [Lad18, LFL16].

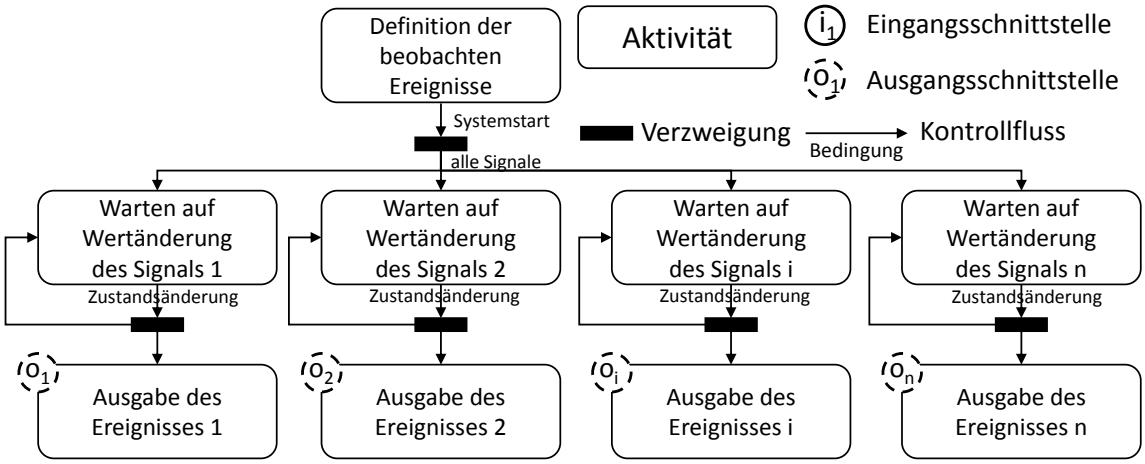


Abbildung 3.7: Artefaktverhalten der Beobachtung eines Bussystems

Über den Kontext werden Ereignisse zudem an einen Besitzer im CPPS gebunden. Dieser Besitzer kann das gesamte CPS sein, aber auch nur eine seiner Komponenten, wie beispielsweise einzelne Maschinen. Dabei haben Komponenten wiederum eine beliebige Anzahl von Elementen, wie beispielsweise Sensoren, die auch Besitzer eines Ereignisses sein können. Der Problemstellung einer minimalen Informationsmodellierung folgend, sollten Kontextinformationen dabei auf möglichst niedrigerer Ebene durch den Nutzer definiert werden. Deshalb ist der Orts- und Typkontext für die Beobachtung von Ereignissen des laufenden CPPS an die unterste Betrachtungsebene von Elementen gebunden, was einem *Bottom-up* Ansatz bei der Informationsmodellierung entspricht. Dabei sind neue Kontexte des Nutzers nur notwendig, wenn das Quellartefakt Ereigniskanäle für neue, bisher nicht im CPS vorhandene Ereignisse bereitstellt. Das Wertetypobjekt der Kontextinformation umfasst dabei mindestens die Ausprägungen Typ-, Orts- und Zeitkontext. Dabei bestehen die Daten des Kontextes aus beliebig vielen *Kontextpaaren*, die eine *Kontextinformation* beschreiben, und dem *Kontextwert*. Kontextpaare sind genau einem Besitzer, wie beispielsweise einem Signal, zugeordnet.

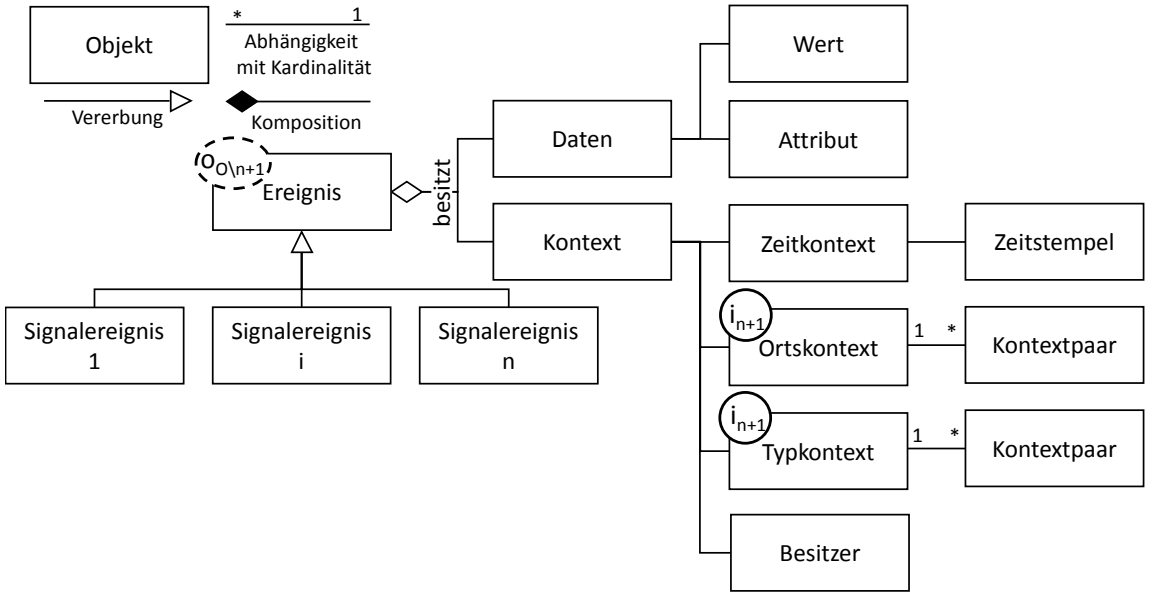


Abbildung 3.8: Datenstruktur eines Ereignisses auf dem Bussystem

Aus den Informationen des Quellartefakts und den Kontextinformationen resultiert die Datenstruktur der Beobachtung, die in Abbildung 3.8 gezeigt wird. Ereignisse haben den Wertetyp $T_{Ereignis} \in \{Signalereignis\ 1, ..., Signalereignis\ i, ..., Signalereignis\ n\}$. Als Daten werden die entsprechenden beobachteten Attribut-Werte-Paare und jeweils der Zeitkontext als gemessener Zeitstempel des aufgetretenen Ereignisses, sowie der modellierte Orts- und Typkontext und der Besitzer angegeben. Entsprechend kann ein Ereignis auch als ein Ereignistupel $(attribut, wert, zeitkontext, ortskontext, typkontext, besitzer)$ betrachtet werden. Der Stream zwischen dem Quell- und Informationsartefakt ergibt sich aus der Sequenz der transferierten Ereignisobjekte, wobei jedes Objekt einem Ereignis auf dem Bussystem entspricht.

Diese Ereignisse werden in Informationsartefakten verarbeitet. Im Fall von Bussystemen ist im Informationsartefakt eine Bestimmung des Zustands über die Zeit vorgesehen. Für die Evolutionsunterstützung müssen Zustände deterministisch und beobachtbar sein, was für die in dieser Arbeit betrachteten CPPSe gegeben ist. Dabei entspricht der Zustand des Artefakts dem Zustand aller seiner Eingangsschnittstellen, der sich immer dann ändert, wenn ein verändertes Ereignis über den korrespondierenden Kanal wahrgenommen wird. Entsprechend beschreibt das Zustandstupel $(wert^1, ..., wert^2, ..., wert^n)$ den momentanen Zustand des CPPSs, wobei $wert_i$ jeweils der Wert des letzten Ereignisses der Eingangsstelle i_i ist.

Grundsätzlich dient die Bestimmung des Zustands im Informationsartefakt dem Wissenstransfer von Informationen, wie er für CPPSe notwendig ist, um eine konsistente Weiternutzung zu ermöglichen [WW18]. Dieser Wissenstransfer wird nach Wang et al. durch die genaue, zeitnahe, konsistente und wertsteigernde Übermittlung von Ereignissen an übergeordneten Datenverarbeitungsartefakte erreicht [WW18]. Die im Informationsartefakte vorhandenen Ereignisse stellen eine Vorstufe von Evolutionswissen über ein CPPS dar, welches auf der dynamischen Interaktion der Hardware und Software des CPPSs basiert und im Sinne von Wang et al. im weiteren Verlauf dieser Arbeit genutzt wird.

Dienstaufrufe:

Im Folgenden werden Aufrufe von Diensten als ein zweites Quellartefakt für die Evolutionsunterstützung beschrieben. Es setzt dabei ein CBCPS voraus, das komponentenorientiert aufgebaut ist. Diese Komponenten stellen Dienste bereit, die über *Methoden* von anderen Komponenten, externen Nutzern oder anderen (Hardware)systemen verwendet werden können. Der Ansatz basiert auf Arbeiten zu Nachrichtensequenzen von Rohr et al. [RvM+08, Roh15] und van Hoorn et al. [vRH+09] und wurde für diese Arbeit entsprechend konzeptionell angepasst, um als Quell- und Informationsartefakte zu fungieren.

Für das Artefaktverhalten müssen zunächst die Ereignisse der zu beobachtenden Dienstaufruf definiert werden. Dafür ist das System entsprechend zu instrumentalisieren indem die entsprechenden Aufrufe beobachtet und in Ereignissen abgebildet werden. Die damit verbundenen Auswirkungen befinden sich in einem Spannungsfeld zwischen dem zusätzlichen Aufwand zur Beobachtung und dem dadurch generierten zusätzlichen Nutzen [Roh15]. Das avisierte Optimum ist dabei von der jeweiligen Anwendung der Methodik und deren Ziel abhängig. In dieser Arbeit werden der Anforderung einer minimalen Beeinflussung folgend, deshalb die beobachteten Aufrufe auf die notwendige Untermenge aller möglichen Aufrufe reduziert. Diese ist im Rahmen dieser Arbeit manuell auf den Start und das Ende aller Dienstmethoden beschränkt worden, um Rückschlüsse auf der Ebene von Aufrufen zu ziehen und dennoch die Kosten der Beobachtung zu begrenzen. Allerdings gibt es auch Methoden zur automatischen Spezifikation der Untermenge, wie der Ansatz von Jung et al. [JHS13].

Abbildung 3.9 zeigt die aus der Beobachtung resultierende Datenstruktur des Quellartefakts. Dabei können die Wertetypobjekte für CBCPSe als Ausprägungen die verschiedenen im CBCPS vorhandenen Methoden haben. Da diese Arbeit im späteren Verlauf auf Performanzmodelle dieser Aufrufe fokussiert, ist hier der Zeitkontext auf Basis der Messdaten der Dienstaufrufen berechnet. Dieser ist durch die jeweiligen Kontextwerte der Startzeit und Endzeit des Aufrufes geben. Zusätzlich ist die Bestimmung des Typ- und des Ortskontextes möglich. Der Typkontext entspricht den beobachteten Methoden. Der Ortskontext entspricht dem Dienst. Diese Kontextinformation könnten auch durch manuelle Kontextwerte ersetzt oder erweitert werden, welche eine höherwertige Abstraktionsebene aufweisen können. Dies wurde allerdings im Fall der Dienstaufrufe vermieden, da bei CBCPSen ein vollautomatisierter Ansatz evaluiert werden soll. Zur Identifikation der Reihenfolge von Aufrufen enthält das Objekt eine Sequenz ID und weitere technische Ausführungsinformationen, welche neben dem Aufrufer (*caller*) und Aufrufenden (*callee*) zur Rückverfolgung und Erstellung von Nachrichtensequenzen dienen (vergleiche [Roh15]). Entsprechend fokussiert der Ansatz nur interne Ereignisse auf Applikationsebene und keine Ereignisse der Umgebung, wie der Cloud-Infrastruktur.

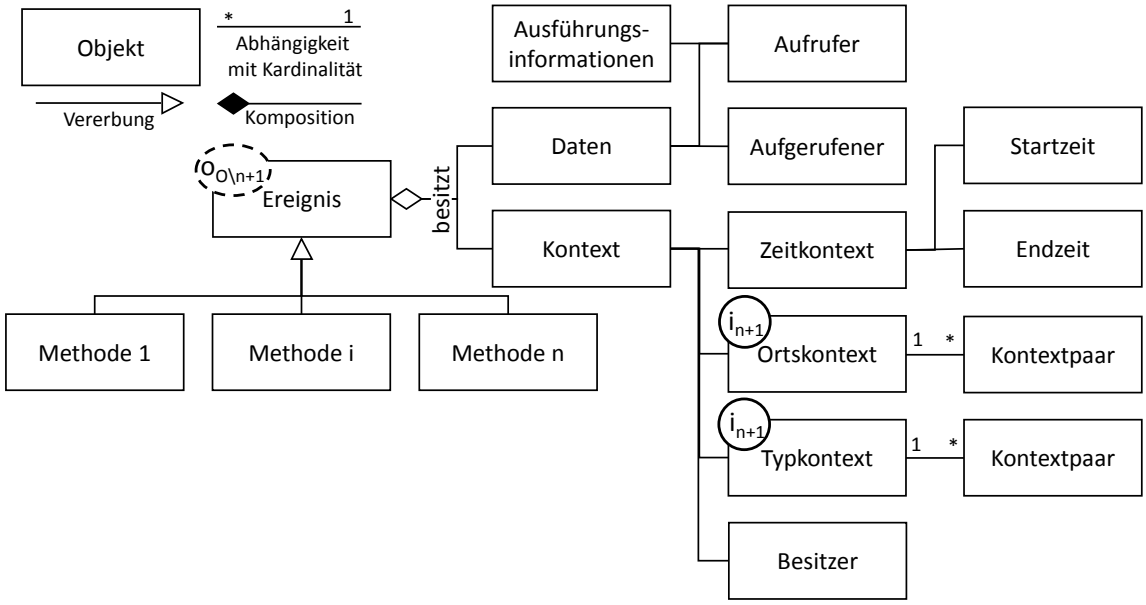


Abbildung 3.9: Datenstruktur eines Dienstaufrufes

Als Aufrufe werden an dieser Stelle nur synchrone Aufrufe verwendet, bei denen die aufrufende Komponente wartet bis der Aufrufende die Anfrage beendet hat. Asynchrone Aufrufe können dabei vorkommen, werden aber von diesem Ansatz zeitlich nicht berücksichtigt. Dies hat den Grund, dass - Rohr [Roh15] folgend - die Zusammengehörigkeit und Reihenfolge der Aufrufe festgelegt werden muss, um die Performanz des CBCPSs zu messen.

Um der Verhaltensfunktion der Artefakte zu genügen muss der Zustand des CBCPSs bestimmt werden. Der Zustand von Aufrufhierarchien bildet, konträr zum Gesamtzustand über die Zeit, einen Zustand ab, der durch Nachrichtensequenzen beschrieben wird. Diese Sequenzen beschreiben, wie die Aufrufe im CBCPS erfolgt sind. Trotz dieser anderen Beschreibungsform ist das Verhalten der Informationsartefakte vergleichbar zu dem für Bussysteme gewählten Vorgehen. Dabei wird allerdings bei der Aktivität der Zustandsbildung eindeutige Sequenzen von Nachrichten im CBCPS mit Hilfe einer Sequenzsynthese rekonstruiert. Die Arbeit verwendet dabei eine entsprechende Methodik von Rohr und van Hoorn et al. [RvM+08, vRH+09]. Die entsprechende softwaretechnische Realisierung wird in Kapitel 4 gezeigt.

Das Resultat der Methodik für Informationsartefakte sind Nachrichtensequenzen.[7] Diese sind eine geordnete Sequenz von Beziehungen zwischen der aufrufenden Methode m_1 und der aufgerufenen Methode m_2. Diese besitzen einerseits einen Typkontext, welcher die Signatur der Methode beschreibt, und andererseits den Ortskontext, in dem die besitzende Komponente und der anbietende Dienst spezifiziert sind. Eine Nachricht wird als Nachrichtentupel $(m_1, m_2, a \in \{Aufruf, Rückgabe\}, zeitpunkt)$ beschrieben. Der erste Aufruf hat dabei den leeren Aufrufenden \$. Die Sequenz ist komplett, wenn er mit diesem startet und endet. Jeder Aufruf muss dabei eine nachfolgende Rückgabe haben, es gibt keine Rückgaben ohne Aufruf und bei jedem direkten Nachfolger ist der vorherige Aufgerufene der nun Aufrufende (siehe [Roh15] für Details). Die Aktion a gibt an, ob es sich um den initialen Aufruf am Anfang der Methode oder um die Rückgabe eines Wertes am Ende des Methodenaufrufes handelt. Zusätzlich erweitert die Arbeit die Nachrichtensequenzen um einen Zeitpunkt, die aus dem Zeitkontext (Startzeit oder Endzeit) abgeleitet wird. Diese Nachrichtensequenzen sind die Zustände, die vom Artefakt als Werteobjekte über den Kanal $c^I_{Zustände}$ transferiert werden.

Grundsätzlich ermöglicht die vorgestellte Methodik zur Beobachtung von Aufrufhierarchien, ein CBCPS anhand seiner getätigten Aufrufe in Artefakten und deren Zuständen (Nachrichtensequenzen) zu repräsentieren. Diese werden in dieser Arbeit genutzt, um Modellartefakte über die Systemarchitektur zu generieren und die Performanz eines CBCPSs zu bestimmen.

3.3.2 Nutzung von assoziierten Entwicklungsartefakten

Neben dem laufenden CPS selbst können auch dessen assoziierte Dokumente betrachtet werden. Diese sind nicht im Verhalten direkt ersichtlich, können aber durch eine Integration in das laufende System verfügbar gemacht werden [Sta17]. Diese Spezifikationen können beispielsweise Entwicklungs- und Testmodelle sein, die während des Entwicklungsprozesses entstanden sind. Ein entscheidender Nachteil dieser Spezifikationen bei der Evolution ist, dass in der Regel keine automatisierte Koevolution erfolgt, was mit dem Einbinden dieser Spezifikationen verbessert werden kann. Dabei können die Informationen der Spezifikationen direkt mit der Evolutionsunterstützung, beispielsweise als Kontextinformationen, verknüpft werden oder die enthaltenden Informationen werden indirekt über eine zusätzliche Methodik, wie eine Simulation oder Analyse, extrahiert und dann als Artefakt eingebunden.

Die operative Ausführung des CPSs verbleibt für diese Arbeit die Hauptquelle von Informationen, aber dennoch stellt der folgende Abschnitt beispielhaft zwei weitere in dieser Arbeit entwickelte Artefakte vor, die ergänzend Ereignisse oder Kontextinformationen über solche Spezifikationen bereitstellen können. Damit wird gezeigt, dass die artefaktbasierte Methodik auch auf andere Bereiche der Evolutionsforschung angewendet werden kann. Dazu können verschiedene Vorgehen adaptiert und integriert werden. Als Beispiel wird zum einen die Verwendung von Testartefakten gezeigt, um Evolutionsinformationen schon vor der eigentlichen Inbetriebnahme zu erhalten. Und zum anderen wird die Gewinnung von Kontextinformationen über die statische Analyse des Quellcodes nach einem etablierten Komponentenmodell für verteilte Systeme gezeigt.

[7] Es wird eine zu Rohr et al. [RvM+08, vRH+09] abweichende Notation von Nachrichtensequenzen verwendet.

Simulation von Testfällen für CPPSe:

Spezifikationen der Entwicklung und Artefakte, die während der Laufzeit verwaltet werden, weisen auf den ersten Blick eine große Ähnlichkeit auf, da beide als Abstraktion von Informationen dienen, indem sie die Struktur oder das Verhalten von Systemaspekten beschreiben. Aber im Detail werden sie aus zwei verschiedenen Motivationen heraus erstellt und damit auch auf Grundlage unterschiedlicher Datenbasen erzeugt: Bei der modellgetriebenen Entwicklung werden Artefakte als eine präskriptive Beschreibung des CPPSs verwendet, indem sie beschreiben, wie das System aufgebaut ist und im Betrieb reagieren soll. Laufzeitartefakte stellen üblicherweise das momentane Verhalten dar und beschreiben das System somit deskriptiv (vergleiche [Küh06]). Dabei basieren Entwicklungsartefakte auf geschätzten und vermuteten Werten, während generierte Artefakte gemessene Werte und Ereignisse verwenden. Die natürliche Schlussfolgerung daraus ist, dass Artefakten der Entwicklung als Startpunkt und Kontextquelle für die spätere Evolution genutzt werden können.

Eine vielversprechende Möglichkeit ist an dieser Stelle die Testphase bei der Inbetriebnahme eines CPSs, in der das gewollte Verhalten überprüft wird. Ein solches vorab spezifiziertes Verhalten ist für die Evolutionsunterstützung von Nutzen, um das Verhalten des CPSs bereits vor Beginn der Evolution abzuschätzen [LFH+15]. Eine Form von Spezifikationen sind dabei Testfälle, die in der Regel für jede Art von Softwaresystem vorliegen. Deshalb wird nachfolgend ein im Rahmen der Forschungstätigkeiten zu dieser Arbeit entwickelter Ansatz vorgestellt, der ein Quellartefakt ermöglicht, welches Ereignissequenzen aus modellbasierten Softwaretestfällen erzeugt. Dies ermöglicht die Extraktion von Wissen direkt aus Entwicklungsartefakten, die dann im Rahmen der Evolution weiterverwendet werden können. Zusätzlich können dadurch Unterschiede zwischen dem reinen Softwareverhalten und dem gemeinsamen Hard- und Softwareverhalten aufgezeigt werden.

Der Ansatz basiert bezüglich der verwendeten Testmodelle auf einer Methode des modellbasierten Testens von Lochau et al. [LBL+14], die nachfolgend kurz erläutert wird. Das modellbasierte Testen ermöglicht dabei eine automatische Generierung von Testfällen auf der Basis von Testmodellen des Verhaltens [UL06]. Ein Testmodell, beispielsweise eine endliche Zustandsmaschine, spezifiziert alle beabsichtigten Verhaltensweisen einer Software mittels beeinflussbarer Eingänge und erwarteter beobachtbarer Ausgänge. Dabei sollen die erzeugten Testfälle die durch Testziele vorgegebenen Abdeckungskritieren (z.B. alle Zustände der Software) erfüllen. Testfälle sind hierbei als Pfade zu interpretieren, die repräsentative Läufe des CPPSs in einer Folge von Testschritten abbilden. Bei der Testausführung wird dann beobachtet, ob unter den stimulierten Eingaben des Pfades, die erwartete Ausgabe erzeugt werden.

Modellbasiertes Testen stellt somit ein **zusätzliches Quellartefakt** dar, das als eine Alternative zum operativen Beobachten eines CPPSs genutzt werden kann. Dazu wird das Verhalten des CPSs simuliert indem Testfälle und ihre Testschritte ausgeführt werden und die Ereignisse dieser Ausführung beobachtet werden. Abbildung 3.10 zeigt das dafür entwickeltes Verfahren. Hierbei müssen zunächst die vorhandenen Testmodelle interpretiert werden. Dabei können unterschiedliche Ausdruckssprachen für Testmodelle gewählt werden. Im gezeigten Verfahren wird dies für CPPSe anhand von sequentiellen Funktionsdiagrammen gemäß des Standards IEC 61131 [DIN09] durchgeführt. Die Ablaufsprache verwendet dabei, vergleichbar mit den beobachteten Ereignissen auf dem Bussystem, Ereignisse von Sensoren und Aktoren zur Beschreibung des Zustands im CPPS. Ein Testschritt ist dabei immer durch ein Paar von Eingangs- und Ausgangsereignissen beschrieben, wobei die Eingangsereignisse durch die Software kontrolliert werden und die Ausgangsereignisse das erwartete resultierende Verhalten des CPPSs repräsentieren.

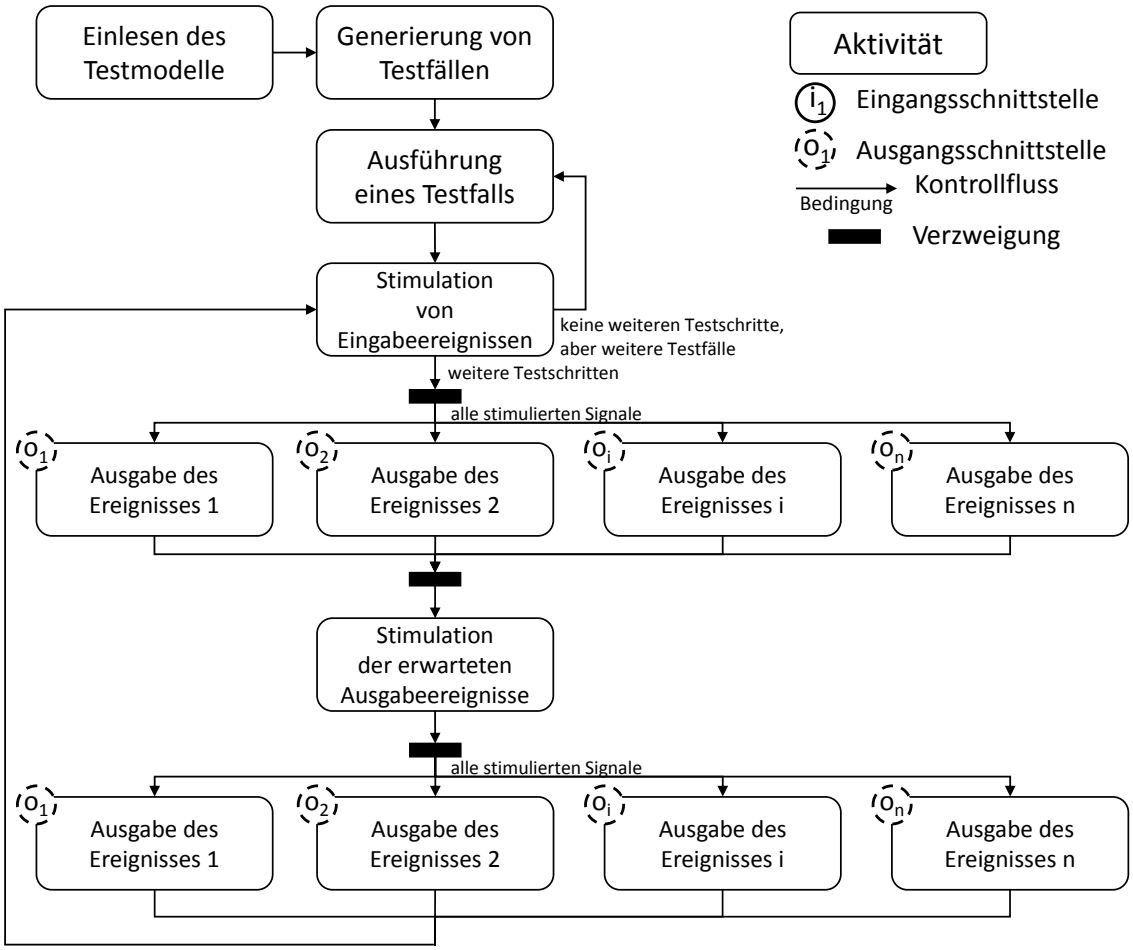


Abbildung 3.10: Artefaktverhalten bei der Simulation von Testfällen

Mithilfe der modellgetriebenen Testfallgenerierung werden so Testfälle erzeugt, die dem spezifizierten Abdeckungskritierum entsprechen. Für die gezeigte Methode sollte dabei zumindest jedes Verhalten einmal oder, für ein stabiles Zeitverhalten, mehrfach beobachtbar sein. Für sequentielle Funktionsdiagramme sollten somit alle Schritte (*steps*), die Eingangsereignisse, jede Transition (*transition*), die Ausgangsereignisse und ihre Verbindungen (*directed links*) mindestens einmal durchlaufen werden. Während der Testfallausführung werden für jeden Testschritt die jeweiligen Ein- und Ausgänge stimuliert. Für jeden stimulierten Ein- und Ausgang werden Ereignisse erzeugt, die durch die Informationsartefakte verarbeitet werden können. Alle Testfälle werden sequenziell ausgeführt, wobei diese Testfälle in der Regel von einer immer gleichen initialen Ausgangslage ausgehen.

Es ist wichtig zu betonen, dass in den verwendeten Testfällen nur die Software getestet wird und die tatsächliche Hardware der Anlage dabei nicht berücksichtigt wird. Entsprechend stimmen diese Testfälle für CPPSe, wegen nicht berücksichtigter physischer Effekte, nicht vollständig mit dem Verhalten des realen CPS überein, obwohl sie die physische Komponente (explizit oder implizit) abbilden. Allerdings beruht diese Abbildung auf Annahmen, die möglicherweise nicht wahr sind, oder keine wirklichen Zeitangaben widerspiegeln [LFH+15]. Dennoch ermöglicht es das so entworfene Quellartefakt mit dem bereits für Bussysteme erläuterten Informationsartefakt bereits vor der eigentlichen Inbetriebnahme das grundlegende Verhalten über Zustände zu erfassen. Dadurch wird durch dieses Quellartefakt ein Transfer von Wissen aus der Entwicklungsphase in die Evolutionsphase realisiert.

Statische Analyse des Quellcodes von CBCPSen:

Entgegen der zuvor verfolgten Extraktion von Daten über die Evolution wird im Folgenden ein weiterer Ansatz dieser Arbeit vorgestellt, der eine zusätzliche Kontextquelle realisiert. Der Ort- und Typkontext wurde in den vorherigen Methoden durch den Nutzer bestimmt, was durch die Kontextquelle automatisiert stattfindet. Dafür wird eine für die Arbeit entwickelte Methodik[8] für CBCPSe vorgestellt, die den Quellcode der Software als Kontextquelle etabliert, ohne dass dafür eine Ausführung dieses Quellcodes notwendig ist. Mit einer solchen statischen Codeanalyse können Softwaremetriken oder die abstrakte Syntax der beteiligten Komponenten jeweils extrahiert werden [Kro12]. Ziel ist es dabei, Informationen, die implizit im Quellcode enthalten sind, durch eine statische Analyse für die Evolution nutzbar zu machen.

Im Allgemeinen werden bei der statischen Codeanalyse verschiedene Metriken genutzt oder kombiniert, um Architekturelemente eines Systems zu identifizieren. Die hier verwendeten Verfahren setzt einen engen Fokus und fokussiert auf CBCPSe, die mit dem *Aktiven Komponententenparadigma* umgesetzt sind (siehe Abschnitt 2.3.4). Für das verwendete Verfahren werden Komponenten, Dienste und ihre Methoden aus Annotationselemente und Methodensignaturen des Quellcodes als Kontextinformation extrahiert. Dadurch kann eine vollständige Generierung von Modellen für die bereits vorgestellten Dienstaufrufsequenzen erreicht werden, indem auch Orts- und Typkontexte von Aufrufen automatisiert bestimmt werden. Das Verfahren wurde im Rahmen dieser Arbeit für die Fallstudien der Evaluation entwickelt, wobei vergleichbare Verfahren, wie die von Walter at al. [Wal18, WSKK17], in einem ähnlichen bzw. gegebenenfalls differenzierteren Artefakt der Kontextquelle resultieren würde.

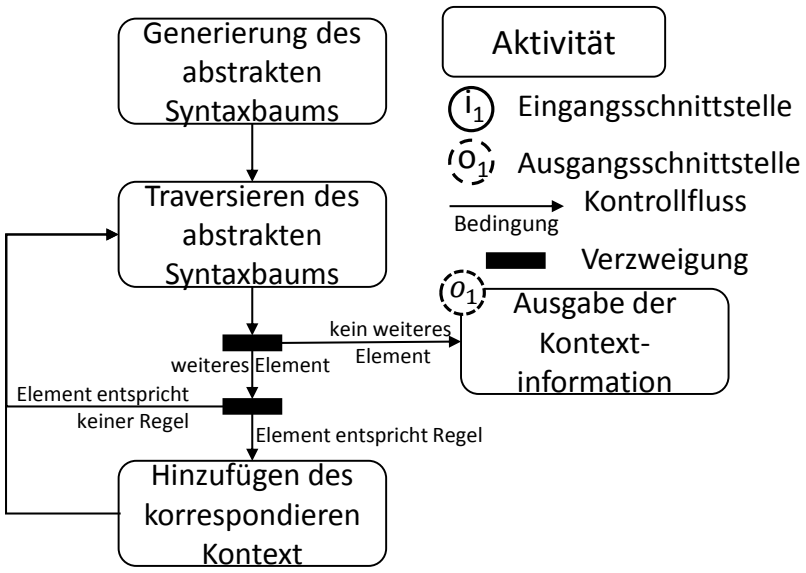


Abbildung 3.11: Artefaktverhalten der statischen Analyse des Quellcodes

Das Verhalten der Kontextquelle, welche die Methodik realisiert, ist in Abbildung 3.11 dargestellt. Dabei wird zunächst der Quellcode eingelesen, um die typischen Elemente, wie Klassen, Variablen oder Methoden zu bestimmen. Das Ergebnis hieraus ist eine formale Repräsentation in hierarchischen abstrakten Teilabschnitten. Dieser abstrakte Syntaxbaum ist dann geeignet, eine statische Codeanalyse durchzuführen.

[8] Die Methodik wurde in Zusammenarbeit mit Nour-Eddine Dadssi im Rahmen einer Masterarbeit [Dad17] entwickelt

Der abstrakte Syntaxbaum wird für die Analyse traversiert. Während des Traversierens werden die Codeelemente entlang von Regeln gefiltert. Die Regeln beschreiben eine Abbildung von Codeelementen, die im abstrakten Syntaxbaum gefunden werden können. Tabelle 3.2 zeigt diese Regeln in Form einer textuellen Beschreibung. Insgesamt werden dabei sieben verschiedene Regeln verwendet: Zum einen werden Komponenten identifiziert, diese werden im Quellcode über die *@Agent*-Annotation von *Aktiven Komponenten* identifiziert. Wenn eine Annotation gefunden wird, entspricht der Namenskonventionen für *Aktive Komponenten* folgend, der Name der Klasse dem Komponentennamen. In einer zweiten Identifikationsregel werden auch die Dienste über die *@Service*-Annotation identifiziert. Diese Dienste bieten Methoden im CBCPS an. Sowohl Komponenten als auch Dienste sind somit Ortskontexte für aufgerufenen Methoden. Durch die Analyse der zusätzlichen *@ProvidedServices*- und *@RequiredServices*-Annotation der Komponenten, werden neben der bloßen Identifizierung von Komponenten und Diensten auch deren Verbindungen extrahiert. Dadurch wird bestimmt, von wem Methoden aufgerufen werden können und zu welcher Komponente sie gehören. Zusätzlich wird der Typkontext der Methoden mit deren Signaturen (Name, Rückgabetyp und Parameter) abgebildet. Diese werden durch die Methodendeklaration des identifizierten Dienstes aus dem Quellcode extrahiert.

	Regelbeschreibung	**Codeelement** **Ortskontext** **Typkontext**
1.	Identifizierung einer Komponente des CPSs	@Agent Annotation $(komponente = komponentenName)$ -
2.	Identifizierung eines Dienstes	@Service Annotation $(dienst = dienstName)$ -
3.	Angebotene Dienstverbindung der besitzenden Komponente	@ProvidedServices{...} Annotation $(angeboteneDienste = \{dienstname_1, ...\})$ -
3.	Benötigten Dienstverbindung der besitzenden Komponente	@ProvidedServices{...} Annotation $(benötigteDienste = \{dienstname_1, ...\})$ -
5.	Methodenbeschreibung der Dienstmethode	Methodendeklaration - $(name = methodenName)$
6.	Rückgabewert der Dienstmethode	Methodendeklaration - $(rückgabe = klassenTyp)$
7.	Parameter der Dienstmethode	Methodendeklaration - $(parameter = \{klassenTyp_1, ...\})$

Tabelle 3.2: Regeln für Korrespondenzen zwischen Codeelementen und Wertepaaren der Kontextinformationen von Dienstmethoden

Wenn kein weiteres Element beim Traversieren vorhanden ist, werden die Kontextinformationen für jede Methode vom Kontextquellartefakt über den Kanal $c^K_{KontextInfo}$ an das Informationsartefakt transferiert, welches diese zur Zustandsbildung nutzt. Mit dem Kontextquellartefakt wird somit eine erweiterte Methode in das Artefaktverhalten einer artefaktbasierten Evolution integriert werden, dass Kontextinformationen, in diesem Fall für Dienstaufrufe, automatisiert extrahiert.

Zusammenfassung der Informationsgewinnung in Quell- und Informationsartefakten

In diesem Abschnitt wurden die Artefakte der Informationsgewinnung vorgestellt, was insbesondere die Quell- und Kontextartefakte umfasst. Für diese Artefakte wurden vier Methoden vorgestellt, die die entsprechenden Verhaltensfunktion der Artefakte jeweils erfüllen. Dabei wurde für CPPSe die Verwendung von Bussystemen und Testfällen vorgestellt. Für CBCPSe wurden Ereignisse von Dienstaufrufen als Ereignisquelle und der Quellcode als Kontextquelle präsentiert. Mit Informationsartefakt wurde gezeigt wie daraus Zustände in Form von Gesamtzuständen von CPPSen und Nachrichtensequenzen von CBCPSen für die weitere Evolutionsunterstützung zusammengesetzt werden können. Dies ermöglicht konsistente Zustände des Verhaltens von CPPSen im Vorfeld und während der operativen Phase über methodische Artefakte bereitzustellen.

Es sei darauf hingewiesen, dass für die gezeigten Methoden in der Literatur eine Reihe von alternativen Vorgehen existieren. Die hier vorgeschlagenen operativen Verfahren eine externe Beobachtung und besitzen dadurch Grenzen bezüglich der zu extrahierenden Informationen. Diese gewählte Grenze ist in der Annahme dieser Arbeit begründet, dass eine externe *Black-box*-Beobachtung eine bessere Akzeptanz für eine Evolutionsunterstützung und eine bessere Übertragbarkeit auf CPSe bietet. Nichtsdestotrotz ist der einheitliche Rahmen der Artefaktstruktur auch für andere interne und auch spezifischere und ggf. komplexe Methoden geeignet.

3.4 Koevolution in Modellartefakten

In der industriellen Praxis ist generell ein Mangel an formalen Modellen sowohl während der Evolution, als auch schon bei der Entwicklung und Konstruktion von industriellen Anlagen zu beobachten [FL00]. Entsprechend geht der vorgestellte Ansatz zur Evolutionsunterstützung über die Erzeugung von Zuständen hinaus. Dafür soll die zweite Hypothese des methodischen Abschnitts überprüft werden, die eine Koevolution von Modellartefakten vorschlägt. Zur Beantwortung dieser Hypothese werden einerseits für die Darstellung und Ableitung von Eigenschaften von Änderungen entwickelte Modelltypen vorgestellt und es wird andererseits überprüft, inwieweit vorhandene Modelltypen dafür genutzt werden können.

Im artefaktbasierten Vorgehens sind Modellartefakte definiert, die durch unterschiedliche Artefaktverhalten und Modelltypen realisiert werden können. Abbildung 3.12 zeigt die dafür erweiterte Artefaktstruktur der Arbeit. Um Modelltypen zu integrieren, müssen Modellartefakte die Verhaltensfunktion $V^{Modellartefakt} : \overrightarrow{\{i_1^I, i_2^O\}} \rightarrow \overrightarrow{\{o_1^M, o_2^M\}}$ für entsprechende Modelltypen erfüllen. Dabei ist die Wissensgrundlage durch die Eingangsschnittstelle i_1^I des Kanals $c^I_{Zustände}$, durch den die Zustände der im vorherigen Abschnitt 3.3 beschriebenen Informationsartefakte transferiert werden, gegeben. Bei CPPS-Modellen sind dies die Gesamtzustände auf dem Bussystem über die Zeit, welche durch Kontextinformationen angereichert sind. Für CBCPSe sind dies Nachrichtensequenzen, welche durch Kontextinformationen aus der statischen Quelltextanalyse angereichert sind.

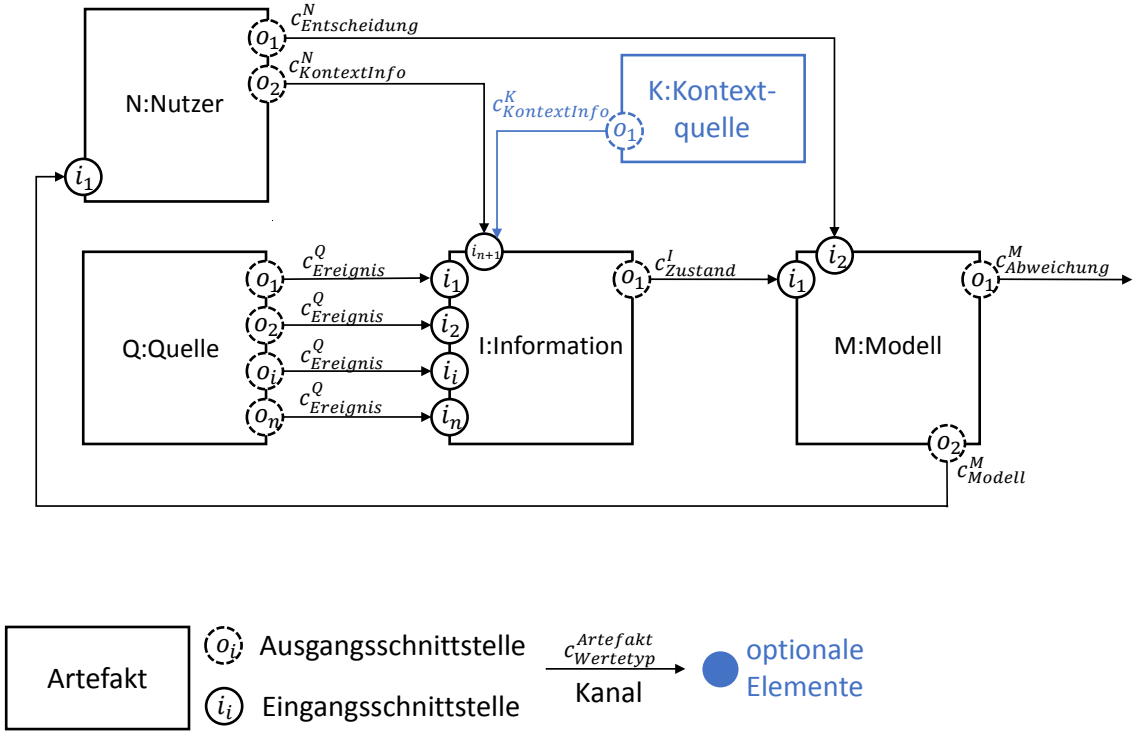


Abbildung 3.12: Artefaktstruktur für die Generierung von Modellartefakten

Dabei muss in beiden Fällen eine Generierung von Modellen und eine Analyse dieser Modelle hinsichtlich spezifizierter (nichtfunktionaler) Eigenschaften möglich sein. Das daraus resultierende Modell wird über den Kanal c^M_{Modell} weiter für die Evolutionsunterstützung bereitgestellt. Daneben müssen die Modelle auch eine modellbasierte Beobachtung der Zustände des CPSs anbieten, damit etwaige Abweichungen dieser Beobachtung durch das Modellartefakt wei-

tergehend zur Verfügung gestellt werden können ($(c^{M}_{Abweichung})$). Diese Informationen werden im weiteren Verlauf für die verschiedenen konzeptionellen Aspekte eines Evolutionsschrittes verwendet. Zusätzlich dazu ermöglicht es die Artefaktstruktur, manuelle Entscheidungen des Nutzers mit einzubinden. Dies hat zum Grund, dass der Nutzer im Sinne des angestrebten halbautomatisierten Ansatzes über die Koevolution der Artefakte mitentscheiden soll. Dies wird insbesondere für CPPS-Modellartefakte verfolgt, die eine Bewertung von Abweichungen durch den Nutzer anstreben. In CBCPS-Modellartefakten werden hingegen alle Änderungen für eine Koevolution automatisch verwendet, um die Koevolution beispielsweise in einem kontinuierlichen Integrationsansatz (*continous integration*) einsetzen zu können.

Im folgenden Abschnitt werden die Struktur und das Verhalten von Modellartefakten betrachtet. An dieser Stelle werden auch die vier in dieser Arbeit mitentwickelten und im weiteren Verlauf verwendeten Modelltypen für CPPSe und CBCPSe vorgestellt. Es ist dabei in der Regel nicht möglich, generelle Modelle oder Methoden zur Generierung und Analyse dieser Modelle für jedes beliebige CPS zu finden (vergleiche [LFL16]). Die Modelle sind entsprechend immer von der Domäne abhängig, weshalb in der modellbasierten Forschung zunehmend domänenspezifische Modellierungssprachen verwendet werden [Keh15]. Entsprechend wurden für CPPSe und CBCPSe jeweils unterschiedliche Modellartefakte eingesetzt.

Im Bereich von Produktionssystemen werden Maschinenzustandsmodelle und Materialflussmodelle für CPPSe verwendet. In Maschinenzustandsmodellen wird der Zustand einer CPPS-Komponente dargestellt, um darauf basierend Kennwerte für diese abzuleiten. In Materialflussmodellen wird insbesondere der Aspekt des unterliegenden Produktionsprozesses in diskreten Fertigungsanlagen betrachtet indem dieser als spezifischer Aspekt herausgegriffen wird. Für CBCPSe werden komponentenorientierte Systeme untersucht. Dafür wurden vorhandene Komponentenmodelle für die Vorhersage von Eigenschaften der Performanz herangezogen. Dies umfasst zum einen ein Modell für Komponenten und ihre Schnittstellen mittels dessen die Systemarchitektur eines CBCPSs beschreiben werden kann und zum anderen ein ergänzendes Modell, das Zeitverhalten von Dienstaufrufen ausdrücken kann.

3.4.1 Allgemeines Modellartefakt

Generell können Modelltypen zwei verschieden Zwecken dienen: Zum einen der Darstellung von spezifischen Eigenschaften eines CPSs, wie der Performanz oder seiner Kopplung. Solche Modelle dienen hauptsächlich der Analyse des CPSs. Zum anderen können Modelltypen die Zusammenhänge zwischen Elementen des CPSs beschreiben und Aussagen dazu machen, wie sich das CPS verhält.

Dabei sind für Modellartefakte einige Einschränkungen notwendig: Zunächst müssen alle in den Artefakten verwendeten Modelle ausschließlich auf den Ereignissen bzw. einer Teilmenge dieser Ereignisse basieren, um den über das Informationsartefakt angereicherten Zuständen zu genügen. Daran anschließend muss auch das Verfahren zur Generierung dieser Modelle ausschließlich aus dem inhärenten Wissen der Ereignisse bzw. den zusätzlichen Kontextinformationen stammen. Die dazu verwendeten Verfahren werden im weiteren Verlauf als *Generierungsverfahren*[9] bezeichnet. Dieses setzt auch voraus, dass das CPS als ereignisbasierte System modelliert sowie dessen Verhalten durch diese Ereignisse ausreichend beschreiben werden kann. Dabei sollte das CPS ein wiederkehrendes bzw. zyklisches Verhalten aufweisen, damit eine Beobachtung ein möglichst komplettes und fehlerfreies Verhalten aufzeichnen kann.

[9] Je nach Anwendungskontext der hier verwendeten Verfahren, werden diese in der Literatur auch als Lernverfahren, Reengineering oder Extraktion bezeichnet. Die Arbeit bleibt dennoch bei dem allgemeinen Begriff der Generierung von Modellen.

Denn nur unter diesen Voraussetzungen kann erreicht werden, dass das Verhalten in einem abstrahierenden Modell beschrieben werden kann, um damit den unterliegenden Prozess des CPSs in einer aussagekräftigeren Form als den Ereignissen zu dokumentieren. Zusätzlich bleiben die Annahmen der Quell- und Informationsartefakte auch für Modellartefakte erhalten.

Es gibt eine Vielzahl von Generierungsverfahren, die in der Evolutionsunterstützung dieser Arbeit eingesetzt werden könnten (siehe verwandte Arbeiten in Abschnitt 5.2), wobei anschließend nur die verwendeten Verfahren vorgestellt werden. Für CPPSe wurde im Rahmen der Forschungstätigkeit umfangreiche Verfahren entwickelt, die zusammen mit Jan Ladiges entwickelt wurden und in seiner Dissertation [Lad18] ausführlich beschrieben sind. Für CBCPSe wurde ein Verfahren speziell für verteilte Systeme auf Basis von *Aktiven Komponenten* entwickelt. Dieses nimmt Anleihen aus den Arbeiten von Walter [Wal18], Krogmann [Kro12] und Brosig et al. [BKK09] und wurde im Rahmen dieser Arbeit auf die Evolutionsunterstützung angewendet. Dabei ist die Methodik zur Evolutionsunterstützung und auch die Systemarchitektur dieser Arbeit nicht auf diese Modelle beschränkt, sofern die notwendigen Bedingungen erfüllt und die notwendigen Verfahren für diese Modelltypen vorhanden oder entwickelt werden.

Dabei sind nach der Taxonomie von Ducasse und Pollet [DP09] die verwendeten Modelle wie folgt zu klassifizieren: Die Modelle haben selbst eine *Dokumentation* und *Analyse* zum Ziel. Über den Evolutionsunterstützungsprozess wird zusätzlich eine halbautomatisierte *Koevolution* und Unterstützung bei der *Durchführung von Evolution* ermöglicht. Dabei wird für die Generierung ein *Bottom-up* Ansatz mit *quasi-automatischen Methoden* auf Basis von *dynamischen Informationen* inklusive einiger weniger *statischer Informationen* verwendet.

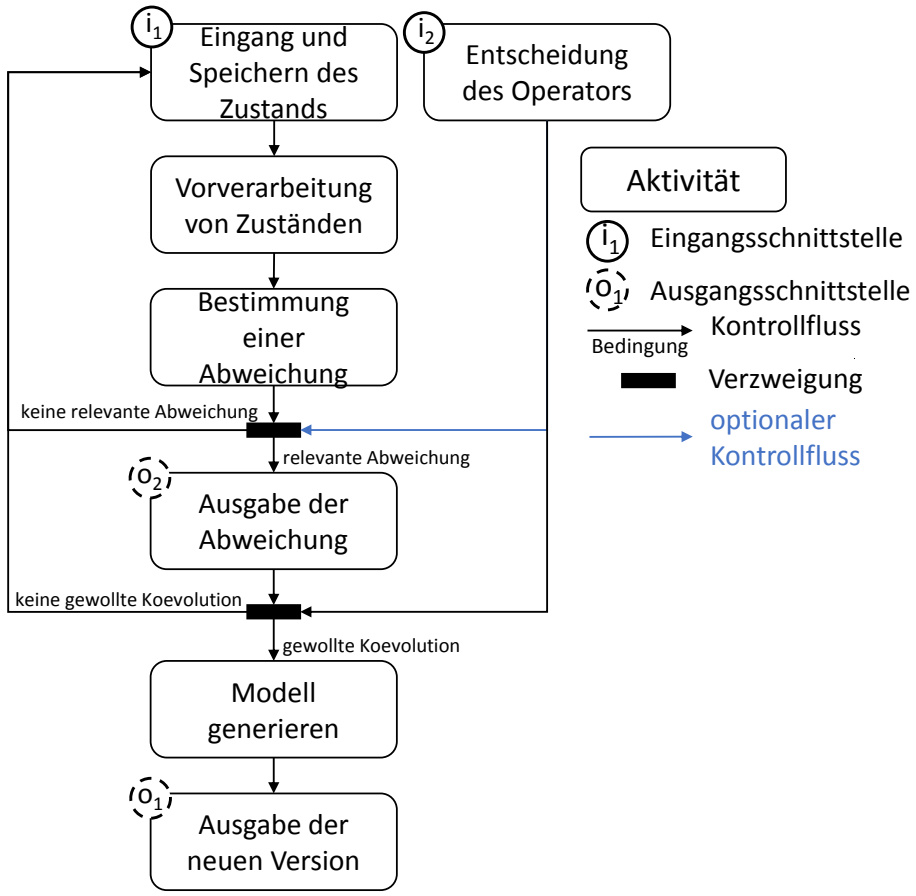


Abbildung 3.13: Allgemeines Artefaktverhalten für Laufzeitmodelle

Für diese Modelle zeigt Abbildung 3.13 ein allgemeines Artefaktverhalten. Bei diesem ist nach dem Eingang eines veränderten Zustandes zunächst zu ermitteln, ob eine Änderung im momentanen Zustand durch das neue Wertetypobjekt des Informationsartefakts vorliegt. Dies muss nicht immer der Fall sein, da beispielsweise auch ein schon beobachteter Aufruf im CPS stattgefunden haben kann, der keine Zustandsänderung zur Folge hat. Anschließend sollte die Veränderung bezüglich des dem Artefakt inhärenten Modells bewertet werden. Dies kann, muss aber nicht mit einer Entscheidung durch den Nutzer unterstützt werden. An dieser Stelle ist zu klären, ob eine Änderung des Zustandes für das Modell von Bedeutung ist. Wenn ein relevanter Zustand dem Modell widerspricht, ist eine Abweichung gegeben, was dann allen mit der Ausgangsschnittstelle o_2 verbundenen Artefakten mitgeteilt wird.

Nun muss entschieden werden, ob diese Abweichung zu einer Koevolution des Artefakts führen soll. Da die vorgestellte Methodik der Evolutionsunterstützung keine automatisierte Entscheidungsarchitektur anstrebt, werden CBCPS-Modelle immer koevolviert und bei CPPSen bleibt es an dieser Stelle bei der Entscheidung des Nutzers. Dieser Nutzer bewertet, ob es sich bei der Abweichung um eine gewollte oder ungewollte Abweichung handelt. An dieser Stelle wäre der Ansatz prinzipiell zu einer Selbstadaption fähig, wenn der Eingangskanal $c^N_{Entscheidung}$ nicht durch einen Nutzer, sondern ein entsprechendes, zur Selbstadaption befähigtes Artefakt, bedient wird. Für die Arbeit wird dies im Ausblick in Abschnitt 7.3 diskutiert.

Anschließend wird die Koevolution des Modells durchgeführt. Dabei gibt es je nach dem verwendeten Verfahren eine Vorverarbeitung der gegebenen Zustände, auf deren Basis dann ein entsprechendes Modell generiert wird. Abgeschlossen ist das Artefaktverhalten, wenn eine neue Modellversion generiert ist und durch die Ausgangsschnittstelle transferiert wurde. Entsprechend entstehen zwei Rückflüsse an den Nutzer des CPS: Zum einen die Modelle, welche eine aktuelle Dokumentation der Evolution zur Verfügung stellen und mit deren Hilfe eine Analyse[10] bezüglich der Eigenschaften des CPSs möglich ist und zum anderen eine Erkennung von Abweichungen (*annomaly detection*), also eine Identifizierung von einem Verhalten, das der abstrakten Beschreibung des vorigen Modelles nicht entspricht.

Die Datenstruktur des Wertetyps besitzt als Daten ein Modellobjekt, welches abhängig von der Modellausprägung ist und einem Meta-Modell folgt. Als Kontext wird das Modell einem Besitzer im CPS zugeordnet und es erhält eine Beschreibung des Modells, inklusive einer versionsbeschreibenden Identifikation.

Die verwendeten Ausprägungen von Modellartefakten werden Folgenden entlang des allgemeinen Modellverhaltens vorgestellt. Dazu wird die Wahl der Modelltypen motiviert und die Modelltypen anhand ihrer Meta-Modelle vorgestellt. Anschließend werden Vorverarbeitungsschritte, Änderungserkennung und Generierungsverfahren für die Modelle gezeigt. Beispiele der Modelle und ihrer Koevolution werden im Rahmen der Evaluation in Kapitel 6 gezeigt.

3.4.2 Materialfluss- und Maschinenzustandsmodelle für CPPSe

Generell gibt es verschiedene Verfahren zur Generierung von CPPS-Modellen aus Ereignissen. In der Domäne von Produktionssystemen sind dies häufig Zustandsautomaten, Markow-Ketten oder Petri-Netze [LFL16]. Im Kontext der Forschungsarbeiten dieser Arbeit wurden spezifische Modelle auf Basis von Petri-Netz-Netzen entwickelt. Ein Petri-Netz ist dabei ein bipartiter Graph, der den Zustand eines CPPSs durch Knoten (Plätze) und Zustandsübergänge durch Kanten des Graphen (Transitionen) beschreibt.

[10] Die Analyse wird im Rahmen der Erstellung von Evolutionsschritten erläutert.

Materialflussmodell (MFPN)

Materialflussmodelle (MFPN)[11] bilden den Materialfluss innerhalb eines Fertigungssystemen, als ein Anwendungsgebiet von CPPS, ab. Die Modelle können genutzt werden um den zugrunde liegenden Fertigungsprozess zu dokumentieren und beispielsweise bezüglich produktionsspezifischer Kennzahlen zu analysieren [LFA+15].

Dabei sind formale Modelle des Materialflusses besonders während der Evolution bedeutend, da durch sie Änderungen in der Systemnutzung durch eine (automatische) Prozessanalyse erkannt werden können [DG92, Tan12]. So haben Materialflussmodelle die Besonderheit einen spezifischen Aspekt des betrachteten CPPSs, nämlichen den Produktionsprozess, darzustellen.

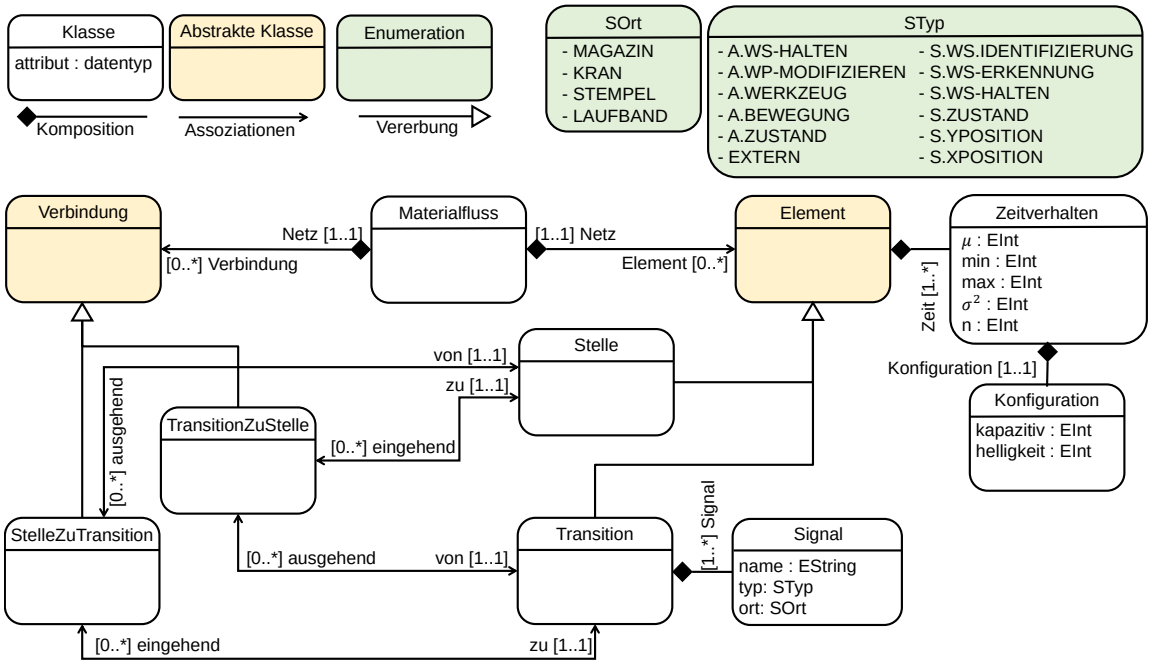


Abbildung 3.14: Meta-Modell für Materialflussmodelle

Abbildung 3.14 zeigt das Meta-Modell zur Modellierung von Materialflüssen, die aus Ereignissen aus einem CPPS generiert werden. Das Meta-Modell besteht dabei aus Elementen, die *Transition* oder *Stellen* sein können. Transitionen entsprechen einem zu beobachteten Wertetyp von Ereignissen und werden mit *Signalen* annotiert, die den Ereigniskanälen des Quellartefaktes entsprechen. Diese Signalelemente haben als Attribute die *Orts- und Typkontexte* der entsprechenden Ereignisse, sowie, sofern vorhanden, ein *Namensattribut*. Die Enumeration halten alle möglichen Ausprägungen der Kontextinformationen und sind im Meta-Model der Fallstudie der Evaluation für CPPSe entsprechend dargestellt. Stellen beschreiben Zeitpunkte, an denen kein weiteres Ereignis beobachtet wird. Beide Elemente haben

[11] Die folgende zwei Modelle für Produktionsanlagen wurden zusammen mit Jan Ladiges im Rahmen des Forschungsprojekts FYPA^{2}C entwickelt. Die Modelle werden in dieser Arbeit verwendet. Die Arbeit gibt allerdings nur einen Kurzüberblick über die dabei entwickelten Modelle. Eine ausführliche Behandlung dieser Modelle und ihrer abgeleiteten Eigenschaften finden sich in entsprechenden Veröffentlichungen der dazugehörigen Ansätze [LHFL14a, HWB+13], der Materialflussmodelle [LFA+15] und der Maschinenzustandsmodelle [LHFL15], der abgeleiteten Eigenschaften [LFHL13, LWA+13], sowie der Dissertation von Jan Ladiges [Lad18]. Die Meta-Modelle und die weiteren Verfahren dieser Arbeit wurden aufbauend dazu für diese Arbeit entwickelt.

zusätzlich das *Zeitverhalten* als zeitlichen Kontext.[12] Dieser ist durch μ, σ, min, max und n gegeben und an die Konfiguration eines Werkstücks gebunden. Für die verwendete Fallstudie entspricht dies beispielsweise dem Material eines Werkstücks. Das Material ist dabei durch seine *Kapazitivität* und *Helligkeit* definiert. Elementen sind weiterhin durch Verbindungen im Materialfluss verbunden. Verbindungen beschreiben den Übergang zwischen Transitionen und Stellen (*TransitionZuStelle*) bzw. zwischen Stellen und Transitionen (*StelleZuTransition*). Dabei haben sowohl Verbindungen Assoziationen auf die Elemente (*eingehend* und *ausgehend*), als auch Elemente eine Assoziationen auf ihre Verbindungen (*von* und *zu*).

Dem allgemeinen Artefaktverhalten folgend, wird die Generierung nach dem Verfahren von Ladiges et al. [LFA+15] durchgeführt, das im Rahmen dieser Arbeit mitentwickelt wurde und nachfolgend kurz erläutert wird; Details finden sich in [Lad18, LFA+15, LFL16]:

Zunächst wird der Stream des Informationsartefakts für Bussysteme verwendet und bezüglich der relevanten Ereignisse gefiltert. Um diese zu bestimmen, werden Ereignisse bezüglich des Typkontextes durch Regeln separiert, wobei auch kombinierte Ereignisse verwendet werden. Entsprechend werden hierbei nur Ereignisse betrachtet, die den Materialfluss abbilden. *Kombinierte Ereignisse* beschreiben dabei mehrere Ereignisse, welche eine Und-Verknüpfung besitzen und von binären Signalen des CPPSs stammen. Ein kombiniertes Ereignis hat den Wert 1, wenn alle Ereignisse den Wert 1 haben. In allen anderen Fällen hat das kombinierte Ereignis den Wert 0. Die Regeln für Materialflussmodelle sind spezifisch auf den verwendeten Typkontext der Ereignisse definiert aber unabhängig von dem unter dieser Definition modellierten CPS.

Als weiterer Schritt müssen die Ereignisse der Signale Werkstückinstanzen zugeordnet werden. Das heißt, es muss identifiziert werden, welche Ereignisse durch die gleichen Werkstücke und welche durch unterschiedliche Werkstücke ausgelöst werden. Dies ist notwendig, da Materialflüsse auf einzelne Werkstückinstanzen definiert sind. Dazu wird die Zeitstabilität zwischen den Ereignissen untersucht. Wenn die gleichen Ereignisse mehrfach in einer stabilen Zeitdifferenz getriggert werden, werden diese denselben Werkstückinstanzen zugeordnet.

Ein besonderes Problem bezüglich der Beobachtung von physischen Aspekten eines CPSs stellen dabei geringe Schwankungen in der Zeitmessung für Ereignisse dar (siehe auch Evaluation in Kapitel 6). Deshalb wird für alle Zustände, auf denen das Modellartefakt operiert, ein Zeitintervall im Informationsartefakt verwendet. Alle Ereignisse, die innerhalb dieses Zeitintervalls auftreten, werden in einem gemeinsamen Zustand zusammengefasst. Dadurch wird eine Erhöhung der Modellkomplexität durch ungenaue und streuende Zeitkontexte vermieden. Es muss jedoch berücksichtigt werden, dass dieses Zeitintervall vom betrachteten Prozess abhängt und dadurch Dynamiken unterhalb des Zeitintervalls nicht erfasst werden.

Da für die Änderungserkennung bereits ein Materialflussmodell vorhanden sein muss, wird zunächst das Generierungsverfahren erläutert. Unter den im allgemeinen Artefaktverhalten genannten Anforderungen einer fehlerfreien Beobachtung in einem endlichen Zeitraum können die Werte von Ereignissen bezüglich ihrer Zeitstabilität in den Zustandstupeln untersucht werden. Dadurch ist festzustellen, welche auslösenden Ereignisse durch die gleichen Werkstücke ausgelöst wurden. Hierbei werden einzelne Transportvorgänge explizit gemacht indem jedes auslösende Ereignis einem Werkstück zugewiesen wird. Durch dieses Zuweisen von Identifikatoren kann bestimmt werden, wie ein Werkstück den Produktionsprozess durchläuft. Der Methodik liegt dabei die Annahme zu Grunde, dass Transportzeiten nur sehr gering

[12] Die Arbeit bildet hier Zeitannotation von TransitionZuStelle-Verbindungen, die nach Ladiges [Lad18] für die Eigenschaftsanalyse bei mehreren Eingangstransitionen notwendig sind, durch mehrere (unterscheidbare) Annotation an der Stelle ab.

schwanken und sich im betrachteten Ortsabschnitt nicht überholen. Entsprechend wird diese Zuordnung zunächst auf eine technische Komponente begrenzt und diese werden dann später zusammengeführt.

Aus diesen Werkstückinstanzen wird danach jeweils ein Petri-Netz generiert. Dazu wird für jede Werkstückinstanz gemäß ihrer Reihenfolge von steigenden Flanken eine Verkettung von Transitionen und Stellen erstellt. Anschließend werden die Sequenzen von Transitionen verglichen und Stellen, welche eine gleiche Vortransition aufweisen, zusammengefasst. Bei mehreren Stellen entscheidet die Ähnlichkeit der weiteren Sequenz. Abschließend werden Transitionen mit denselben Vor- und Nachstelle zusammengefasst und Mehrdeutigkeiten aufgelöst. Zuletzt wird aus den Identifikationssignalen das Konfigurationselement erzeugt und aus dem Zeitkontext das Zeitverhalten abgeleitet. Das Meta-Modell kann dabei mehr Petri-Netze ausdrücken, als dies durch das Materialflussmodell vorgesehen ist oder durch das Generierungsverfahren erreicht werden kann. Allerdings können alle generierten Modelle durch das Meta-Modell ausgedrückt werden. Dies hat den Grund, dass das Meta-Model dadurch in seiner Komplexität verringert wird.

Das generierte Materialflussmodell kann dann gegen alle durch das Informationsartefakt transferierten Zuständen geprüft werden. Damit können Abweichungen des neu beobachteten Verhaltens gefunden werden. Um diese Abweichungen zu erkennen, wird das aktuelle Modell immer entsprechend den auftretenden Ereignissen im Stream seiner Eingangsschnittstelle simuliert. Dazu wird für jedes relevante Ereignis die Markierung durch das Feuern der dazugehörigen Transition durch das Materialflussmodell geschaltet. In diesem Ansatz herrscht Konsistenz zwischen dem beobachteten CPPS und dem Modell genau dann, wenn die beobachteten Ereignisse, die im Zustand des Informationsartefakts sichtbar sind, durch eine korrekte Feuersequenz des Petri-Netzes des Materialflussmodells abgedeckt ist.

Dem Artefaktverhalten entsprechend muss dann entschieden werden, ob die entdeckte Abweichung gewollt oder ungewollt ist und ob es dementsprechend einer Koevolution des Modells bedarf. Diese Entscheidung wird jeweils durch den (menschlichen) Nutzer gefällt. Nur bei einer gewollten Evolution bedarf es einer Koevolution im Modellartefakt. Diese wird dadurch erreicht, dass das Generierungsverfahren erneut ausgeführt wird.

Zusammenfassend lassen sich mit den Materialflussmodellen der Materialfluss eines CPPSs bzw. im Speziellen von Fertigungssystemen abbilden. Zusätzlich lassen sich Abweichungen bezüglich des laufenden CPPS feststellen sowie das Materialflussmodell entsprechend des beobachteten Verhaltens generieren. Dies genügt, um dem allgemeinen Modellartefakt zu genügen und damit in die Evolutionsunterstützung eingebunden zu werden.

Maschinenzustandsmodell (MSPN)

Ziel von Maschinenzustandsmodellen (MSPN) ist es, eine formale Spezifikation des dynamischen Verhaltens innerhalb einer CPPS-Komponente zu dokumentieren. Maschinenzustandsmodelle versuchen dabei das komplette beobachtbare Verhalten zu beschreiben und eine geringe Komplexität und Permissivität zu erreichen, ohne zu viel an Allgemeingültigkeit einzubüßen.

Das Meta-Modell für Maschinenzustandsmodelle, welches in Abbildung 3.15 gezeigt wird, hat eine große Ähnlichkeit zu dem Meta-Modell von Materialflussmodell. Dabei enthält es genauso Elemente (*Transition* und *Stellen*) und Verbindungen (*TransitionZuStelle* und *StelleZuTransition*). Entgegen dem Modell für Materialflussmodelle sind bei Maschinenzustandsmodellen allerdings Elemente in ihrem Zeitverhalten nicht bezüglich der Konfiguration von Werkstücken separiert. Denn es wird an dieser Stelle keine Erkennung von Werkstücken sondern die

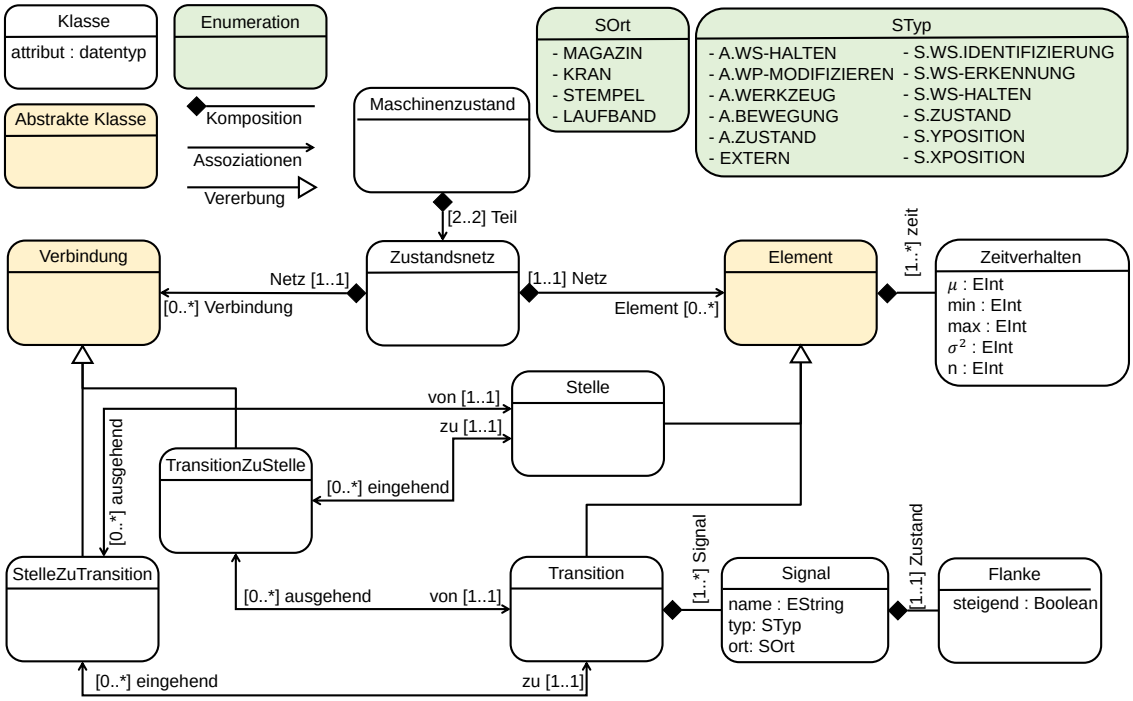


Abbildung 3.15: Meta-Modell für Maschinenzustandsmodelle

Zustände von Komponenten betrachtet. Dementsprechend können Ereignisse der Signale in Maschinenzustandsmodellen sowohl eine fallende als auch eine steigende *Flanke* aufweisen. Der Zustand des Signalelements ist durch das Attribut *steigend* der Flanke gekennzeichnet. Dieses ist wahr, wenn es sich um eine steigende Flanke handelt und unwahr, wenn es eine fallende Flanke beschreibt. Weiterhin besteht das Maschinenzustandsmodell aus einem maschinenbezogenen Teil und einen signalbezogenen Teil.

Das notwendige Generierungsverfahren nach Ladiges et al. [LHFL15], welches im Rahmen dieser Arbeit mitentwickelt wurde, wird folgend kurz erläutert; Details finden sich in [Lad18, LHFL15, LFL16]: Für Maschinenzustandsmodelle wird nur eine Teilmenge der verfügbaren Ereignisse der beobachteten Zustände verwendet. Hierbei wird der Ortskontext der Ereignisse genutzt, um das CPS zu partitionieren. Dies ist notwendig, um das Modell in seiner Komplexität beherrschbar zu machen und damit den Dokumentationscharakter zu erhalten.

Das Generierungsverfahren erzeugt für jede technische Komponente ein Maschinenzustandsmodell aus den Zustandstupeln und ihren zeitlichen Abhängigkeiten. Dabei besteht die Generierung aus drei Schritten:

Im ersten Schritt wird der maschinenbezogene Teil des Verhaltens der technischen Komponente abgebildete indem eine Kausalitätsspezifikation nach Lefebvre et al. [LL11] erzeugt wird, die für Maschinenzustandsmodelle erweitert wurde (siehe [LHFL15]). Dazu wird zunächst eine Matrix abgeleitet, in der alle verursachenden Ereignisse der Zustandstupel bezüglich ihrer direkten Nachfolgeübergänge geordnet sind. Aus dieser Matrix wird eine Inzidenzmatrix berechnet, die jede Nachfolgerbeziehung durch eine Kante abbildet. Abschließend wird diese Inzidenzmatrix in ihrer Komplexität reduziert indem Stellen zusammengeführt werden. Weiterhin wird eine Anfangsmarkierung, die den aktuellen Zustand des CPPSs darstellt, berechnet und der Dokumentationscharakter durch eine Duplizierung von Transitionen erhöht.

Im zweiten Schritt wird mit dem signalbezogenen Teil der Unteranpassung des Maschinenzustandsmodells entgegengewirkt. Dabei besteht ein Spannungsfeld zwischen der Eingrenzung der erlaubten Zustände des CPPSs und der Generalisierung des Modells [LL11, VRV+10]. Um eine Balance in diesem Spannungsfeld zu finden, wird die Tatsache ausgenutzt, dass Ereignisse von binären Signalen in ihrer Wertemenge beschränkt sind. Durch diese Abbildung der beschränkten Wertemenge werden die möglichen Zustände verringert, aber die Generalität wird dabei nur bedingt eingeschränkt [Lad18]. Somit werden für jedes Signal jeweils eine Stelle für die fallende und steigende Flanke erzeugt. Diese werden dann mit den dementsprechenden Transitionen verbunden.

Als letzter Schritt werden Zeitinformationen an die Transitionen des maschinenbezogenen Teils annotiert. Dazu wird aus allen Zustandstupeln, die Ereignisse mit dem (kombinierten) Signal der Transition haben, die entsprechenden Attribute des Zeitverhaltens abgeleitet.

Die Erkennung von Abweichungen und die darin definierte Konsistenz ist analog zu Materialflussmodellen, wobei mit Maschinenzustandsmodellen Abweichungen im Verhalten der Zustände einer technischen Komponente erkannt werden. Dazu werden im Artefaktverhalten die gefilterten Zustände parallel zum simulierten Maschinenzustandsmodell untersucht. Eine Entscheidung über die Absicht der Evolution wird auch bei Maschinenzustandsmodellen durch den Nutzer getroffen.

Zusammenfassend beschreiben Maschinenzustandsmodelle den Zustand einer Komponente eines CPPSs bzw. im Speziellen von Fertigungssystemen und erfüllen über das allgemeine Modellartefakt die Verhaltensfunktion von Modellartefakten.

3.4.3 Palladio Komponentenmodelle für CBCPSe

Unter dem Begriff der Architekturentdeckung (*architecture discovery*) werden Methoden subsumiert, die Informationen über die Softwarearchitektur eines Systems extrahieren. Eine Form von extrahieren Elementen sind Komponenten und ihre Dienste. Um allerdings die Qualität eines Systems während der Evolution zu untersuchen, müssen nicht nur seine funktionalen Eigenschaften überprüft werden, sondern es müssen auch die nichtfunktionalen Eigenschaften, wie z.B. die Performanz, mit betrachtet werden [RBB+].

Das Palladio Component Model (PCM) ist eine domänenspezifische Modellierungssprache für komponentenbasierte Softwarearchitekturen, die speziell auf die Beschreibung der Performanz spezialisiert ist. In dieser Arbeit wird PCM deshalb beispielhaft als Modellartefakt genutzt, das für ein CBCPS die Dokumentation der Systemarchitektur und die Ableitung von Eigenschaften der Performanz, wie u.a. Antwortzeiten oder Durchsatz, erlaubt.

Das Meta-Modell von PCM ist in mehrere domänenspezifische Modellierungssprachen unterteilt, die auf Entwicklerrollen für komponentenbasierte Systeme abgestimmt sind. In dieser Arbeit werden zwei dieser Modelltypen betrachtet. Beide werden in dieser Arbeit für die Sichtweise von Modellartefakten adaptiert und es wird ein Generierungsverfahren zu deren Erstellung aus Nachrichtensequenzen vorgestellt.

Repositorymodelle:

Es wird zunächst das **Repositorymodell** von PCM vorgestellt, welches die Assoziationen von Komponenten und Diensten[13] beschreibt. Diese statische Struktur spezifiziert, welche Dienste angeboten und benötigt werden, und hilft den Entwicklern das System zu verstehen.

[13] PCM spricht entgegen der Terminologie dieser Arbeit hier von Schnittstellen.

In Abbildung 3.16 wird der für Arbeit relevante Ausschnitt des Meta-Modells für PCM-Repositorymodelle gezeigt. Dabei werden zum besseren Verständnis Modellelemente auf die Terminologie dieser Arbeit übertragen, sowie einige, für andere Teile der PCM-Modelle relevanten, abstrakten Oberklassen, die komplexere Komponententyphierarchie, sowie die konkrete Ausprägung von Datentypen vernachlässigt. Die vollständigen, der Terminologie von PCM entsprechenden Meta-Modelle, sind durch Reussner et al. [RBB+, BKR09] veröffentlicht. Das übergeordnete Modellelement *Entität* beschreibt die *Komponenten* und *Dienste* des *Repositorymodells*, sowie das Modell selbst. Dabei kann das Modell aus beliebig vielen *Komponenten* und *Diensten* bestehen. Dienste werden von Komponenten angeboten sowie benötigt, wodurch die Komponenten die entsprechende Rolle annehmen. Beide Rollen sind dabei nicht beschränkt. Dienste sind durch ihre *Signatur* gekennzeichnet. Dies sind einerseits der *Methodenname* und andererseits *Parameter* und *Rückgabewerte*, die beim Aufruf verwendet werden. Beide binden sich an die Datentypen des Repositorymodells, was entweder primitive Datentypen, Collections oder komponierte Datentypen sind (nicht dargestellt in Abbildung 3.16).

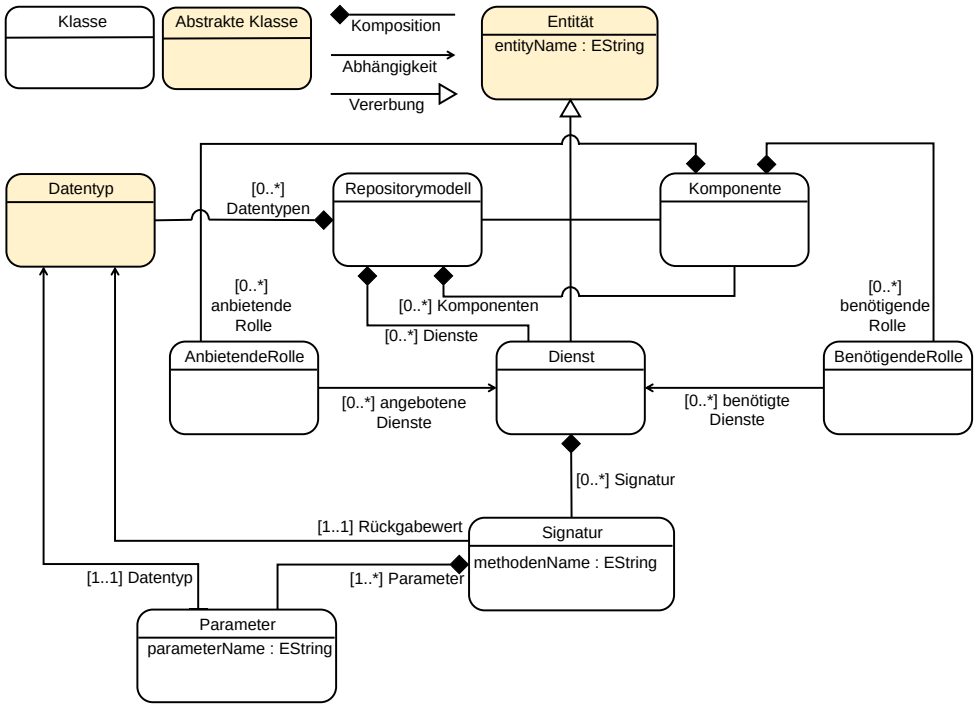


Abbildung 3.16: (Angepasster) Ausschnitt des Meta-Modells von PCM-Repositorymodellen (nach [RBB+, BKR09])

Für die Generierung dieser Modelle wurde im Rahmen dieser Arbeit ein einfaches Generierungsverfahren entwickelt, um diese mit allen für die Evolutionsunterstützung notwendigen Information aus den Nachrichtensequenzen und ihren Kontextinformationen abzuleiten. Die Repositorymodelle basieren dabei zum überwiegenden Teil auf den Kontextinformationen der Methoden. Folglich ist auch keine Vorverarbeitung, wie bei den vorgestellten CPPS-Modellen notwendig. Die hierzu entwickelte Methodik ist in der Verfahrensvorschrift 1 in Pseudocode angegeben. Dabei wird der Stream von Nachrichtensequenzen abgearbeitet indem für jedes Nachrichtentupel geprüft wird, ob die erforderlichen Komponenten und Dienste sowie Verbindungen und Methoden vorliegen. Entsprechend ist es für das Verfahren unerheblich, ob ein neues Repositorymodell generiert wird, oder ein bereits vorhandenen erweitert. Die Komponenten werden hinsichtlich ihrer angebotenen und benötigten Dienste überprüft. Wenn

die Komponente oder ein Dienst nicht im Modell vorhanden ist, wird diese Entität mit dem entsprechenden, im Ortskontext der Methode angegebenen Methodenname erzeugt. Nach der Erzeugung dieser Entität kann ein Element des Typs *AnbietendeRolle* bzw. *BenötigtendeRolle* mit ihren jeweiligen Assoziationen zur Komponente und dem Dienst erzeugt werden. Zusätzlich wird das Modellelement der Signatur aus dem Typkontext abgeleitet. Dieses beinhaltet den Namen der Methode sowie die Assoziationen und Modellelemente des Rückgabewerts und des Parameters. Der Wert der Kontextpaare gibt dabei den Datentyp im Repositorymodell vor.

Dem Artefaktverhalten für Modellartefakte folgend benötigen Repositorymodell auch eine Möglichkeit zur Bestimmung von Abweichungen. Dazu werden, wie bei den vorgestellten CPPS-Modellen, neu eingehende Nachrichtensequenzen mit dem vorhandenen Modell verglichen. Konsistenz zwischen einer beobachteten Nachrichtensequenz und dem Modell herrscht genau dann, wenn alle Komponenten und Dienste im Modell vorhanden sind und alle Methoden und Verbindungen zwischen diesen modelliert sind. Bei einer erkannten Abweichung wird das Modell mit der Verfahrensvorschrift aktualisiert und damit zum CPS koevolviert.

Mit dem Repositorymodell können somit Komponenten und Dienste durch die Auswertung von Kontexten der Methodenaufrufe abgebildet werden. Sie erfüllen das allgemeinen Modellartefakt und können somit in der Evolutionsunterstützung genutzt werden.

Verfahrensvorschrift 1: Verarbeitung von Nachrichtensequenzen zum Erzeugen eines Repositorymodells

Eingabe: Stream von Nachrichtensequenzen: $\overrightarrow{c^{I}_{Zustände}} : \mathbb{R}_{+} \rightarrow T_{Zustände} \cup \{\square\}$ mit
$T_{Zustände} \in \{$Nachrichtensequenz $nachrichtenArray[\,]\}$;
Nachrichtentupel $nachrichtenArray[i] = (m_1, m_2, a, zeitpunkt)\}$;
Methode $m = (name, rückgabe, parameter, dienst,$
$komponente(angeboteneDienste, benötigteDienste))$

Ausgabe: Repositorymodell *modell*

while *neues* $nachrichtenArray[\,]$ *aus* $\overrightarrow{c^{I}_{Zustände}}$ **do**
  foreach $n \in nachrichtenArray[\,]$ **do**
    /* Ortskontexte für angebotene Dienste setzen */
    foreach $n.angebotenerDienst \in n.angeboteneDienste$ **do**
      if $n.angebotenerDienst \notin modell$ **then**
        füge $n.angebotenerDienst$ zu *modell* hinzu
      if $n.komponente \notin modell$ **then**
        füge $n.komponente$ zu *modell* hinzu
      setzte $AnbietendeRolle(n.komponente, n.angebotenerDienst)$
    /* Ortskontexte für benötigte Dienste setzen */
    foreach $n.benötigterDienst \in n.benötigterDienste$ **do**
      if $benötigterDienst \notin modell$ **then**
        füge $n.benötigterDienst$ zu *modell* hinzu
      if $n.komponente \notin modell$ **then**
        füge $n.komponente$ zu *modell* hinzu
      setzte $BenötigendeRolle(n.komponente, n.benötigterDienst)$
    /* Beobachtete Signatur in Dienst erzeugen */
    if $Signatur(name) \notin dienst$ **then**
      füge $Signatur(name, rückgabe, parameter)$ zu *dienst* hinzu

Rückgabe *modell*

Verhaltensmodelle für Methoden

Für die späte erläuterte Analyse der Performanz reichen die Repositorymodelle nicht aus. Deshalb wird ein zusätzliches PCM-Modell für die Modellierung des Verhaltens von Methoden eines Dienstes[14] verwendet. Diese Verhaltensmodelle für Methoden modellieren den Kontrollfluss der angebotenen Methoden und die Kontexte[15], die diese Methoden benötigen. Dies entspricht damit einem *Grey-box*-Ansatz [RBB+], wobei die Arbeit entgegen dem PCM-Ansatz den Quellcode nicht als Informationsquelle nutzt.

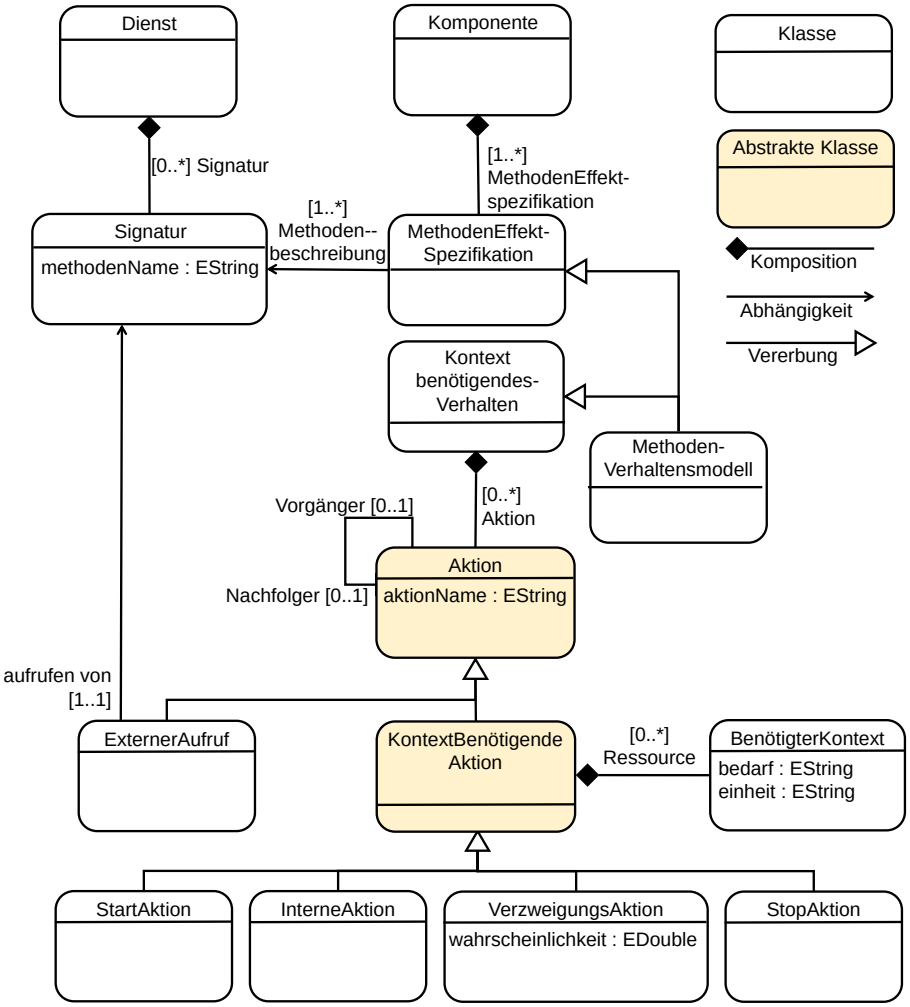


Abbildung 3.17: (Angepasster) Ausschnitt des Meta-Modells von PCM-RessourceDemandingServiceEffektSpecifications (vergleiche [RBB+, BKR09])

Die Verhaltensmodelle werden in PCM in der ResourceDemandingServiceEffectSpecifications (RDSEFF) definiert. Dabei ermöglichen diese Modelle es, Methoden bezüglich ihrer Nutzung von Ressourcen und ihrer genutzten Dienstaufrufe zu spezifizieren. In der vorliegenden Arbeit wird dabei nicht der volle Funktionsumfang dieser Modelle genutzt, da bei der Beobachtung viele Informationen des Quellcodes, die im Kontrollfluss des Modells abgebildet werden, nicht in den Zuständen der Informationsartefakte vorhanden sind. Dabei hat das in PCM verwendete Modell insbesondere Verbindungen zu den Ressourcen der *Assemblierungsmodelle* für die Inbetriebnahme sowie parametrische Abhängigkeiten, die auf Variablen des Quellcodes beruhen.

[14] PCM spricht entgegen der Terminologie dieser Arbeit hier von Services der Schnittstellen.
[15] PCM spricht hierbei von performanzbeeinflussenden Ressourcen

Auf Basis von Informationen aus einer Beobachtung und deren Kontextes können zur Beschreibung des Kontrollflusses und seiner Aktionen allerdings lediglich heuristische Beschreibungen verwendet werden. Deshalb werden Ressourcen, neben den externen Dienstaufrufen, auf ihren extern sichtbaren, zeitlichen Kontext reduziert, wobei der Typ- und Ortskontext bereits durch das Repositorymodell gegeben ist. Das daraus resultierende Meta-Modell von Verhaltensmodellen für Methoden, welches vollständig in Modellelementen eines *RDSEFF*-Modells abgebildet werden kann, zeigt Abbildung 3.17 in der Terminologie der Arbeit.

Ein ***Methodenverhaltensmodell*** besitzt zum einen ein *MethodenEffektSpezikation*-Element und zum anderen ein *KontextbenötigendeVerhalten*-Element. Das erste Element der Spezifikation beschreibt, welche Methode durch das Verhalten definiert wird. Das zweite Element bildet das Verhalten der Aufrufhierarchie und der Kontextnutzung in einem Kontrollfluss ab. Dabei kann eine Spezifikation einerseits eine oder mehrere Methoden beschreiben, es können andererseits aber auch beliebig viele Spezifikationen in der Komponente vorliegen. Der Grund für diese Zweiteilung des Modells ist, dass das Verhalten auch iterativ in sich selbst verwendet werden kann [RBB+].

Der Kontrollfluss wird durch beliebig viele abstrakte *Aktionen* beschrieben, die über eine Vorgänger- und Nachfolgerbeziehung miteinander verbunden sind. Aktionen können den Elementtyp *KontextbenötigendeAktionen* besitzen. Diese beschreiben insbesondere die *InterneAktion*-Elemente der Methode, welche eine bestimmte Zeitspanne zur Ausführung benötigen, die über das *BenötigterKontext*-Element spezifiziert ist. Dieser Zeitkontext wird, wie bei Materialflussmodellen, durch μ, σ, max, min und n beschrieben. Als weitere Aktion existieren externe Dienstaufrufe (*ExternerAufruf*), welche genau eine andere Methode eines Dienstes aufrufen. Die Zeitanforderungen von externen Aktionen werden in dem Methodenverhaltensmodell nicht modelliert, weil das innere Verhalten dieser Aktionen durch die Komponente, welchen den aufgerufenen Dienst anbietet, modelliert wird. Allerdings kann eine Methode mehrere Dienstaufrufe enthalten.

Diese Verknüpfung in einem solchen Kontrollfluss wird durch weitere Aktionen beschrieben. Der Kontrollfluss beginnt immer mit einem *StartAktion*-Element und schließt mit einem *StopAktion*-Element ab. Dazwischen können mehre Aufruf mit Oder-Semantik (Verzweigungs-Aktion)[16] stattfinden, wobei diese zwei neuen Elemente des Typs *KontextBenötigendeVerhalten* beinhaltet. Zusammenfassend besteht der Kontrollfluss einer Methode aus Dienstaufrufen und internen Aktionen, die Zeit verbrauchen und über verschiedene Operatoren verknüpft sind.

Generierungsverfahren für diese Arten von Modellen nutzen zumeist den Quellcode als Hauptquelle für Informationen. Da in dieser Arbeit Informationen der externen Beobachtung und eingeschränkte Modelle genutzt werden, wird dazu ein eigenes *Black-Box*-Verfahren verwendet. In diesen Verfahren wird der interne Kontrollfluss innerhalb einer Komponente als unbekannt angenommen [Kro12]. Da die Ressource Zeit schon durch das Informationsartefakt gegeben ist, verwendet die Arbeit ein Vorgehen, das den Zeitverbrauch anhand des Zeitkontexten der Nachrichtensequenzen approximiert. Da die in dieser Arbeit betrachteten CBCPSe synchrone Aufrufe voraussetzen, können *ExternerAufruf*-Elemente direkt aus den Nachrichtensequenzen ableitet werden und der Zeitverbrauch wird als *InterneAktion*-Element mit dem entsprechenden *BenötigterKontext*-Elementen modelliert.

[16] Eine Und-Semantik sowie Schleifen wurden an dieser Stelle nicht berücksichtigt, da diese unter den Annahmen von synchronen Dienstaufrufen nicht notwendig sind. Und-Semantiken wären aber konzeptionell denkbar und in PCM abbildbar.

Die Generierung von Methodenverhaltensmodellen weist Ähnlichkeit zu der Generierung von Materialflussmodellen für Werkstückinstanzen auf. Beide beschreiben Instanzen von gleichen oder zumindest vergleichbaren verketteten Ereignissen. Dazu muss, wie auch bei den CPPS-Modellen, eine Filterung der Nachrichtensequenzen entsprechend des zu generierenden Modells vorgenommen werden. Dies ist notwendig, da nicht alle Methoden eine vollständige Nachrichtensequenz abbilden, sondern auch nur ein Teil einer Nachrichtensequenz beschreiben können.

Das verwendete Generierungsverfahren zeigt Verfahrensvorschrift 2, welche aus einer Nachrichtensequenz Folgen von Aktionen erzeugt. Um für eine Methode m das erste relevante Nachrichtentupel zu identifizieren, muss in der Nachrichtensequenz ein Tupel gefunden werden, welches einen Aufruf vornimmt, der als zweite Methode m_2, die hier betrachtete Methode besitzt. Wenn ein solches Nachrichtentupel gefunden ist, werden solange Aktionen für Nachrichtentupel erzeugt bis die Rückgabe des Methodenaufrufs beobachtet wird. Dies ist möglich, da die Nachrichtentupel entsprechend der Verarbeitung im Informationsartefakt zeitlich geordnet vorliegen und über die Identifikation der Sequenz keine parallelen Aufrufe möglich sind. Die Verkettung von Aktionen ist sequenziell, wobei diese immer mit einem *StartAktion*-Element beginnen und mit einem *StopAktion*-Element beendet werden. Einzig externe Dienstaufrufe unterbrechen die Verarbeitung der Methode. Wenn ein solcher Aufruf beobachtet wird, der von der betrachteten Methode ausgehen muss, wird die bisher verbrauchte Zeit als *InterneAktion*-Element modelliert. Nach der Rückgabe des Dienstaufrufes wird die Zeit für die Verarbeitung erneut gemessen. Der Aufruf an sich bleibt somit für das Modell dieser Methode unberücksichtigt, da auch der Zeitverbrauch in der jeweils aufgerufenen Methode Berücksichtigung findet. Entsprechend wird bei der Simulation des PCM-Modells an dieser Stelle das Verhaltensmodell der aufgerufenen Methode indirekt verwendet und somit der Zeitverbrauch korrekt abgebildet.

Nach der Generierung der Aktionsketten werden entsprechend des Verfahrens für Materialflussmodellen (siehe [LFA+15]) gleiche Aktionen zusammengefasst. Dadurch entstehen unterschiedliche *Aufrufpfade*, die durch Verzweigungen oder auch die Zusammenführungen mit einem *VerzweigungsAktion*-Element modelliert werden. Diese können auch basierend auf der Anzahl der beobachteten Instanzen, welche den Aufrufpfad in ihrer Sequenz widerspiegeln, mit Häufigkeiten für die Simulation in PCM versehen werden. Als Resultat dieses Vorgehen entsteht das abzubildene Methodenverhaltensmodell.

Die Erkennung von Abweichungen und die darin definierte Konsistenz ist analog zu Repositorymodellen. Dazu werden im Artefaktverhalten die Nachrichtensequenzen mit den in den Methoden modellierten Aufrufpfaden verglichen. Eine Konsistenz besteht, wenn alle externen Aufrufe in mindesten einem Verhaltenspfad abgebildet sind und der Zeitbedarf nicht signifikant von dem modellierten Zeitbedarf in den *InterneAktion*-Elementen abweicht. Wie im Repositorymodell werden auch hier Abweichungen als gewollt angenommen. Methodenverhaltensmodellen können somit das Verhalten beim Aufruf von Dienstmethoden sowie deren zeitlicher Kontext abbilden. Dabei erfüllen sie die Anforderungen das allgemeine Modellartefakt, weshalb sie in der Evolutionsunterstützung genutzt werden können.

Zusammenfassung der Koevolution in Modellartefakten

Generell kann jeder Typ von Modell für eine Dokumentation in Form von koevolvierenden Modellartefakten verwendet werden. Denn nach der Evolutionsunterstützung bestehen diesbezüglich nur einige Einschränkungen der unterliegenden CPSe und deren Ereignisse. Dabei erfüllen die hier vorgestellten Modelltypen über das allgemeine Modellartefakt dessen Verhaltensfunktion $V^{Modellartefakt} : \overrightarrow{\{i_1^I, i_2^O\}} \rightarrow \overrightarrow{\{o_1^M, o_2^M\}}$. Dazu erlauben die Modellartefakte

insbesondere eine Generierung und Abweichungserkennung wodurch eine Koevolution ihres Modells möglich wird.

Denn das bisher vorgestellte Artefaktverhalten ermöglicht, dass Ereignisse und Kontexte aus verschiedenen Quellen extrahiert werden und in Zuständen zusammengeführt werden. Entlang dieser Zustände können Modelle der verschiedenen Modelltypen generiert werden. Eine Generierung wird dabei durch eine erkannte (gewollte) Abweichung angestoßen, woraufhin das Modell mit dem durch die Abweichung erkannten nicht modellkonformen Verhalten aktualisiert wird. Durch diese Koevolution zwischen dem beobachteten Verhaltendem und dem Modell entstehen stetig neue Versionen, die das jeweils zur Erstellungszeit aktuelle Verhalten des CPSs abbilden.

Durch dieses Artefaktverhalten kann die methodische zweite Hypothese einer Generierung und Koevolution in Modellartefakten bestätigt werden.

Verfahrensvorschrift 2: Generierung einer Folge von Kontext-BenötigendenAktionen aus einer Nachrichtensequenz

Eingabe: Nachrichtensequenz $nachrichtenArray[\,]\}$ mit
Nachrichtentupel $nachrichtenArray[i] = (m_1, m_2, a, zeitpunkt)\}$;
Methode $m = (name, rückgabe, parameter, dienst,$
$komponente(angeboteneDienste, benötigteDienste))$

Eingabe: Betrachtete Methode $methode$

Ausgabe: KontextbenötigendesVerhalten $verhalten$

for $i = 0;\ i < |nachrichtenArray[\,]|;\ i^{++}$ **do**
- $(aktion_{letzte}, zeit_{letzte})$ /* letzte erzeugte Aktion mit Zeit */
- m_{erste} /* initial betrachteter Methodenaufruf */
- /* Erster betrachteter Aufruf */
- **if** $methode == n_{Start}.m_2$ **and** $n_{Start}.a == Aufruf$ **then**
  - füge $aktion_{start}$ zu $verhalten$ hinzu
  - $(aktion_{letzte}, zeit_{letzte} = (aktion_{start}, n_{zeitpunkt}$
  - $m_{erste} = m_1$
- /* Externer Methodenaufruf */
- **else if** $methode == n.m_1$ **and** $n.a == Aufruf$ **then**
  - /* Zeitkontext, der durch die Methode verbraucht wurde */
  - berechne Zeit $t = n.zeitpunkt - zeit_{letzte}$
  - füge $aktion_{interne}(BenötigterKontext(\text{Zeit}, t))$ zu $verhalten$ hinzu
  - verbinde $aktion_{interne}$ mit $aktion_{letzte}$
  - füge $aktion_{externerAufruf}((n.m_2.dienst).n.m2)$ zu $verhalten$ hinzu
  - verbinde $aktion_{interne}$ mit $aktion_{externerAufruf}$
  - $(aktion_{letzte}, zeit_{letzte}) = (aktion_{externerAufruf}, n.zeitpunkt)$
- /* Rückhabe des externen Methodenaufrufs */
- **else if** $methode == n.m_2$ **and** $m_{erste} \neq n.m_1$ **and** $n.a == Rückgabe$ **then**
  - $(aktion_{letzte}, zeit_{letzte} = (aktion_{letzte}, zeitpunkt)$
- /* Ende der Betrachtung */
- **else if** $methode == n.m_2$ **and** $m_{erste} == n.m_1$ **and** $n.a == Rückgabe$ **then**
  - füge $aktion_{stop}$ zu $verhalten$ hinzu
  - verbinde $aktion_{letzte}$ mit $aktion_{stop}$
  - **Rückgabe** $verhalten$

3.5 Erstellung von Evolutionsschritten

Die Koevolution von Artefakten und dem CPS, welche im vorherigen Abschnitt behandelt wurde, kann das Verhalten und die Struktur eines CPSs in Modellartfakte abbilden und aktuell halten. Im Folgenden wird die Beschreibung von Evolutionsschritten auf so erzeugte Versionen angewendet, um Änderungen in einem holistischen Evolutionsprozess abzubilden. Mit diesem Abschnitt werden damit Inhalte der ersten Hypothese, einer Darstellung in Evolutionsschritten, mit solchen der zweiten Hypothese, einer Koevolution in Artefakten, verknüpft.

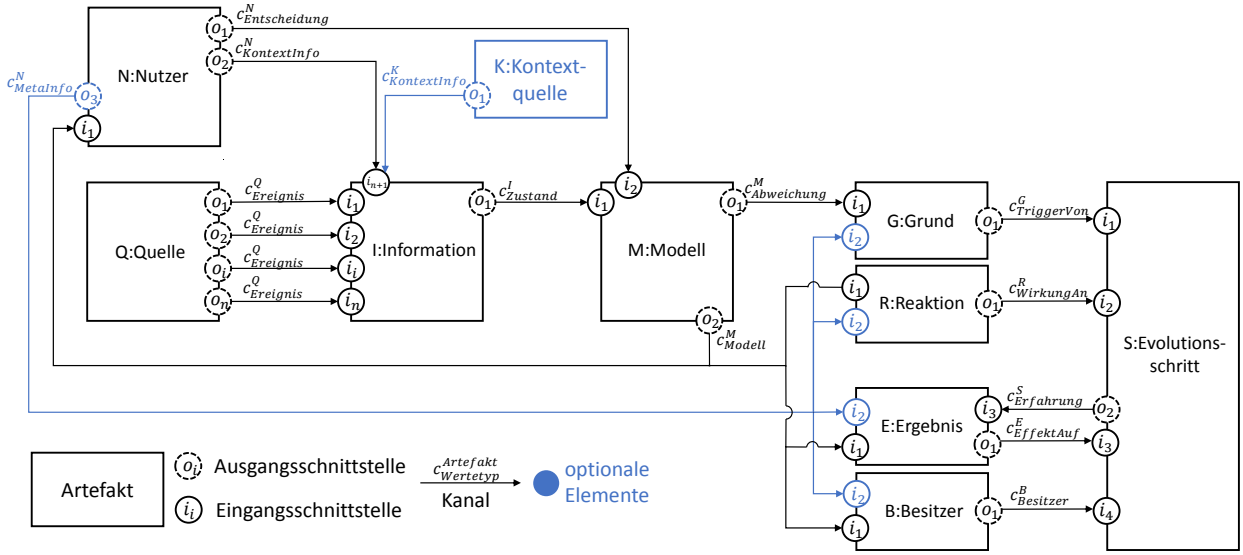


Abbildung 3.18: Artefaktstruktur für die Erstellung von Evolutionsschritten

Abbildung 3.18 stellt die für Evolutionsschritte entwickelten Artefakte dar. Das zentrale *Evolutionsschrittartefakt* folgt der in Abschnitt 3.2.4 entwickelten Beschreibung von Evolutionsschritten. Die einzelnen Aspekte eines Evolutionsschrittes sind dabei ihrerseits Artefakte, die über Kanäle mit dem Evolutionsschrittartefakt verbunden sind und dadurch ihre Informationen für Evolutionsschritte zugreifbar machen. Grundlage sind dabei die drei Dimensionen des CRI-Modells, welche durch das Grund-, Reaktions- und Ergebnisartefakt repräsentiert werden.

Die wichtigste Eingangsschnittstelle des Evolutionsschrittartefakts ist das *WirkungAn*-Wertetypobjekt der Reaktion. Diese Objekte beschreiben die Änderung des Evolutionsschrittes, welche auf ausführbaren, modellbasierten Änderungen beruht (Abschnitte 3.5.1-3.5.3). Zusätzlich zur Reaktion wird der Grund durch das *TriggerAuf*-Wertetypobjekt beschrieben, das auf einer Kategorisierung der Abweichungen zwischen zwei Versionen von Modellartefakten basiert (Abschnitt 3.5.5). Mit dem Ergebnis lassen sich die Auswirkungen auf (nichtfunktionale) Eigenschaften von CPS-Komponenten abschätzen (Abschnitt 3.5.4). Zusätzlich zum CRI-Modell existiert ein *Besitzerartefakt*, welches die Systemteile und den Nutzer des CPSs bezüglich des Evolutionsschrittes identifiziert (Abschnitt 3.5.6).

Alle diese Artefakte verfolgen den im Rahmen des Evolutionsverständnisses bereits erläuterten halbautomatisierten Ansatz und der Unterteilung in Daten und Kontext. Somit können diese Artefakte auch durch Kontextinformationen des Nutzers in den Dimensionen des CRI-Modells ergänzt werden, um die automatisierte Ableitung zu unterstützen.

3.5.1 Reaktion als geliftete Modelldifferenz

Für die Beschreibung einer Reaktion eines Evolutionsschrittes müssen aus den vorhandenen deskriptiven Modellartefakten Änderungen identifiziert und extrahiert werden. Das Softwarekonfigurationsmanagement ist dafür ein geeignetes Forschungsgebiet, da es eine kooperative Verwaltung von Differenzen ermöglicht (siehe Kapitel 5). Die vorliegende Arbeit verwendet deshalb auf das Abstraktionsniveau des Nutzers geliftete Differenzen, um ihn zu unterstützen, Änderungen an einem Modell zu verstehen. Nachfolgend wird zunächst ein Überblick über das Artefaktverhalten und die daraus resultierende Datenstruktur gegeben. In den anschließenden zwei Abschnitten wird dann die Zuordnung von Modellelementen und die Erstellung eines Wirkungsobjektes aus zwei Modellen detaillierter erläutert.

Wie bereits im Grundlagenkapitel festgestellt, das Cyber-Netzwerk ein entscheidendes Merkmal von CPSen. Die daraus resultierende Vernetzung von vorher isolierten und nun vernetzten Komponenten in einem CPS soll deshalb für Evolutionsschritte explizit mit betrachtet werden. Deshalb werden in Evolutionsschritten neben Versionen auch immer Varianten berücksichtigt.

Unter diesen Voraussetzungen zeigt Abbildung 3.19 das in dieser Arbeit verfolgte Vorgehen bei der Identifizierung einer Reaktion. Diese sollen durch das Liften von Differenzen (siehe Abschnitt 2.2.5) zwischen zwei Versionen oder Varianten der generierten Modelle erfolgen. Als Ausgangspunkt dienen verschiedene Modellobjekte eines einzelnen Modellartefakts und damit das Artefaktverhalten $V^{Reaktion} : \overrightarrow{\{i_1^M, i_2^O\}} \rightarrow \overrightarrow{\{o_1^R\}}$ oder Varianten von zwei verschiedenen Modellartefakten und damit das Artefaktverhalten $V^{Reaktion} : \overrightarrow{\{i_1^M, \tilde{i}_1^M, i_2^O\}} \rightarrow \overrightarrow{\{o_1^R\}}$. Dabei hat das Reaktionsartefakt jeweils einen Kanal zu jedem Modellartefakten, welche immer Objekte von einem identischen Wertetyp transferieren. Dies bedeutet, dass Modelle das gleiche Meta-Modell haben, aber unterschiedliche Modellinstanzen sind.

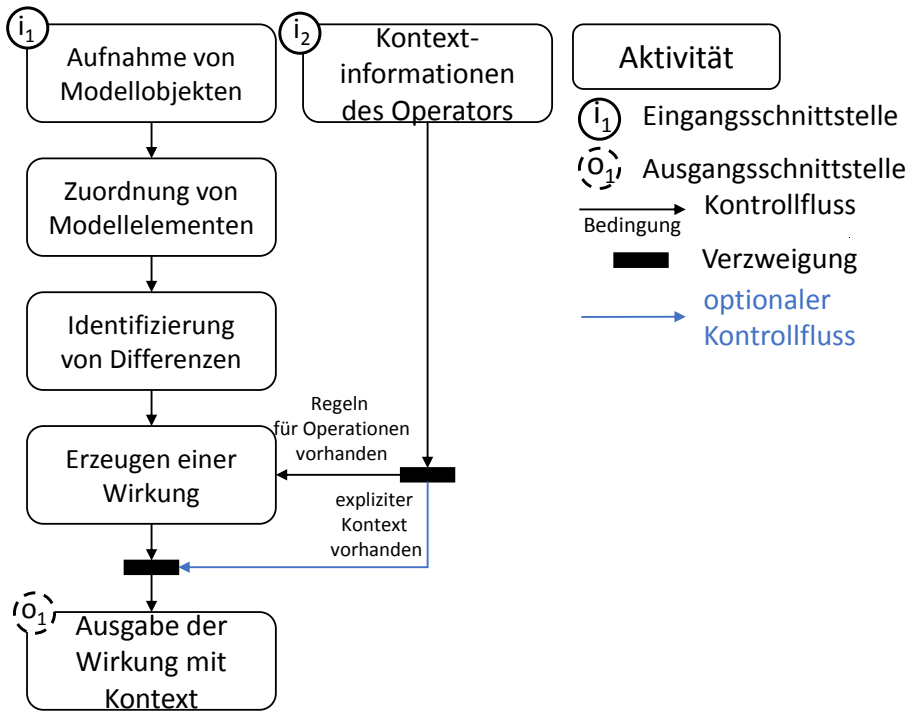


Abbildung 3.19: Artefaktverhalten zur Erstellung eines Streams von Wirkungen als Reaktion eines Evolutionsschrittes

Eine Reaktion soll die vorgenommene Änderung in einem CPS modellbasiert beschreiben. Für die Identifizierung einer solchen Reaktion über Differenzen sind für diese Arbeit drei entscheidende Aktivitäten notwendig: Als erste Aktivität ist es notwendig Modellelemente von zwei Modellen entsprechend ihrer Kontextinformationen einander *zuzuordnen* (Abschnitt 3.5.2). Unter dieser Zuordnung kann als zweite Aktivität eine *Differenz identifiziert* werden, die eine Änderung auf niedrigem Abstraktionsniveau beschreibt. Um diese im Sinne einer schrittbasierten Evolution aussagekräftiger zu machen, werden diese identifizierten Änderungen in einer dritten Aktivität in Operationen zusammengeführt. Auf Basis dieser Operationen wird ein Editierskript generiert, das unter den Voraussetzungen eines Modellkontextes als *Wirkung des Evolutionsschrittes* agiert (Abschnitt 3.5.3).

Zusätzlich zu den Kontextinformationen des Modells sieht der artefaktbasierte Prozess - und damit das Verhalten des Reaktionsartefakts - auch noch optionale, explizite Kontextinformationen vor. Diese werden über die Eingangsschnittstelle des Nutzers i_2 an das Reaktionsartefakt übertragen (vergleiche Abbildung 3.18 und 3.19). Die zusätzlichen Kontextinformationen werden in konkretisierten Kategorien der Reaktion gegeben (siehe Abschnitt 4.2.2), welche nicht automatisch extrahiert, aber für die Wirkung relevant sein können. Beispielsweise können dadurch die abgeleiteten Wirkungen näher spezifiziert werden indem beispielsweise festgehalten wird, dass sich diese Wirkung auf verschiedene Disziplinen wie z.B. Elektrik, Mechanik oder Software auswirkt.

Zusammenfassend besteht das transferierte *WirkungAn*-Wertetypobjekt damit aus den in Abbildung 3.20 dargestellten Objekten. Die Ausprägungen sind abhängig von den Modelltypen. Als Daten enthalten die Wertetypobjekte eine Wirkung des Evolutionsschrittes. Diese besteht aus dem Editierskript der Modelldifferenz und einem Modellkontext, der genutzt wird, um die Anwendbarkeit des Editierskripts zu untersuchen. Ferner kann auch der *Wirkung*-Kontext, sowie der *An*-Kontext durch den Nutzer spezifiziert werden, um diese als Meta-Informationen der Evolutionsunterstützung zur Verfügung zu stellen. Zusätzlich ist der Besitzer als Kontext gegeben.

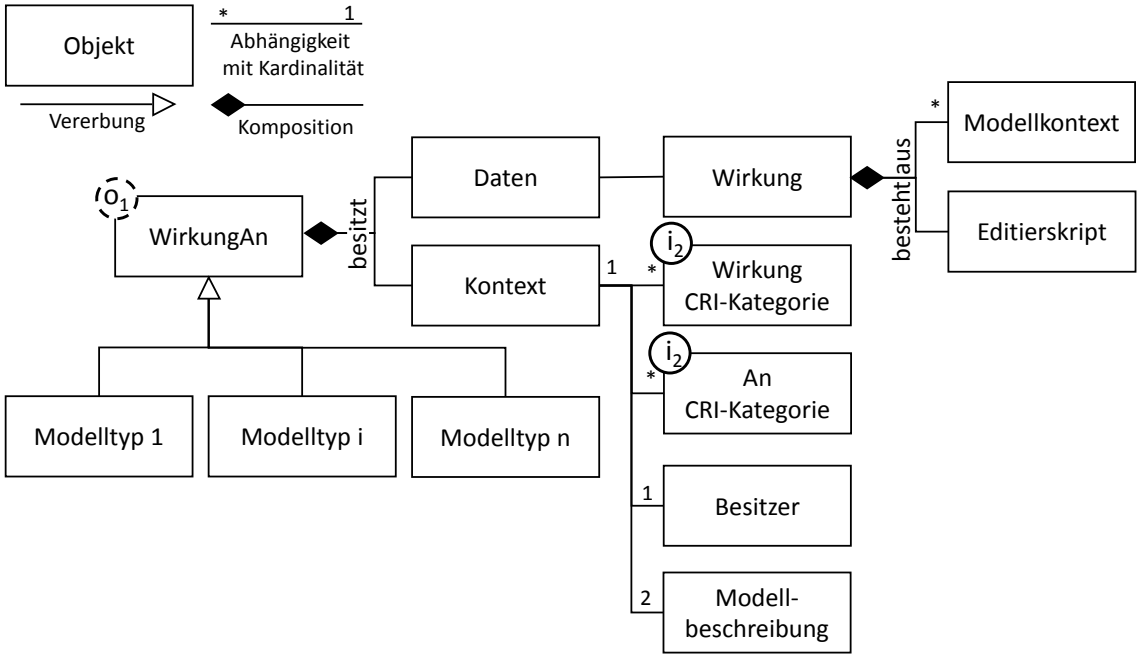


Abbildung 3.20: Datenstruktur des *WirkungAn*-Wertetypobjekts

3.5.2 Zuordnung von Modellelementen

Die Qualität von identifizierten modellbasierten Änderungen hängt stark von der Qualität der **Zuordnung** (*matching*) ihrer Elemente ab [CHFL18]. Deshalb wird nachfolgend die Aktivität der Zuordnung des Reaktionsartefaktes beschrieben. Denn nur, wenn gleiche Elemente identifiziert werden können, kann festgelegt werden, ob bestimmte Elemente durch die Änderung modifiziert, hinzugefügt oder entfernt wurden. Diese Zuordnung von Modellelementen aus zwei Modellen wird nachfolgend bestimmt. Dazu wird zunächst ein allgemeines Verfahren vorgestellt und dann wird dieses Verfahren auf die unterschiedlichen Modelltypen angewendet.

Eine gängige Methodik für die Zuordnung ist der Abgleich persistenter eindeutiger Identifikatoren oder Namen [PKH+18]. Allerdings können solche eindeutigen Identifikatoren im Allgemeinen in erlernten Verhaltensmodellen nicht angenommen werden, insbesondere wenn verschiedene Varianten eines CPSs betrachtet werden. Denn insbesondere CPSe, welche zur Inbetriebnahme auf Hardwarekomponenten angewiesen sind, werden an unterschiedlichen Orten auch in der Regel unabhängig voneinander entwickelt, in Betrieb genommen und evolviert. Entsprechend haben CPSe, selbst wenn sie gleiche Komponenten besitzen, oft unterschiedliche Namen für gleiche Elemente. Zum Beispiel kann ein Sensor zur Detektion in seinem Namen je nach Namenskonventionen seinen Ort (z.B. Kran), seine Funktionalität (z.B. detektieren) oder seinen Typ (z.B. Lichtschranke) enthalten.

Deshalb werden in der modellbasierten Forschung auch ähnlichkeitsbasierte Verfahren eingesetzt, die auch nicht über Namen identifizierbare Modellelemente zuordnen, wenn sie eine hohe Ähnlichkeit aufweisen [KKPS12]. Dazu muss eine **Modellzuordnung**, wie in Formel 3.10 bestimmt werden, die eine partielle, injektive, referenzen- und typerhaltende Zuordnung von Modellelementen vornimmt. Dabei besteht eine Modellzuordnung $Z(m, \tilde{m})$ aus einer Menge von Zuordnungen $e_i \leftrightarrow \tilde{e_i}$ einzelner Modellelementen der beiden betrachteten Modelle m und $\tilde{m}$.

$$Z(m, \tilde{m}) = \{e_i \leftrightarrow \tilde{e_i}, \cdots, e_j \leftrightarrow \tilde{e_j}\} \; mit \; e_i, \; e_j \in m, \; \tilde{e_i}, \; \tilde{e_j} \in \tilde{m} \tag{3.10}$$

Dem Meta-Modell folgend werden nur Modellelemente des gleichen Elements und unter Erhaltung der Referenztypen zugeordnet. Um eine Unterscheidung zwischen den Modellelementen gleichen Elements zu erreichen, ist die semantische Bedeutung der Elemente relevant. Dabei existieren in den Modellen viele Elemente, die semantisch an andere Elemente gebunden sind. Deshalb verwendet die im Rahmen dieser Arbeit vorgeschlagene Zuordnung Ähnlichkeiten von semantisch bedeutenden Elementen und bestimmt die Zuordnung weiterer Elemente durch Abhängigkeiten. Eine solche Abhängigkeit liegt beispielsweise bei Kontrollflusselementen oder Kontextelementen vor, die genau einem semantisch bedeutenden Element, wie z.B. einem Signal des Materialflusses, zugeordnet sind.

Dies bedeutet allerdings nicht, dass diese abhängigen Elemente bei der Ähnlichkeitsbestimmung keine Rolle spielen, denn sie können durchaus bei der Zuordnung der semantisch bedeutenden Elemente verwendet werden, um deren Ähnlichkeit zu bestimmen. Dafür muss ein ähnlichkeitsbasiertes Verfahren für zwei semantisch bedeutende Modellelement $e \in m$ und $\tilde{e} \in \tilde{m}$ eine ***Ähnlichkeitsfunktion*** der Form $f_{\ddot{A}hnlichkeit}(e, \tilde{e}) = g, \; with \; g \in [0, 1]$ bereitstellen. Und zudem wird für alle weiteren Elemente eine Funktion benötigt, welche die Zuordnung mit Hilfe der Relation zu den semantisch bedeutenden Elementen entscheidet.

Ähnlichkeitsfunktionen sind von der Semantik der betrachteten Modelle abhängig. Sowohl die CPPS- als auch CBCPS-Modelle nutzen Sequenzen von Ereignissen, um Modellelemente in Folgen zu verknüpfen. In CPPSen sind dies Transitionen und in CBCPSen Methodenaufrufe. Diese Folgen ändern sich während der Evolution. Dabei würde ein reines ähnlichkeitsbasiertes

Verfahren die Reihenfolge in den Folgen der Modelelemente nicht berücksichtigen. Aus diesem Grund wird nicht nur auf der Grundlage von den Einzelähnlichkeiten der zwei Modellelementen entschieden, sondern zusätzlich wird auch die Reihenfolge in den Folgen berücksichtigt. Dazu werden jeweils zwei Folgen, aus denen die Modelle generiert werden, zueinander angeordnet.

Eine Folge $(e_i)_{i=1,\ldots,n}$ kann formal mit den n Modellelemente $e_1, e_2, \ldots, e_i, \ldots, e_n$ definiert werden. Die Indexe geben dabei an, an welcher Stelle in der Reihe sich das Element befindet. Eine **Anordnung** (*alignment*) ist dann eine Zuordnung von jeweils einem Element e_i der einen Folge zu einem Element der anderen Folge $\tilde{e}_j$. Wobei nicht alle Elemente der Folgen zugeordnet werden müssen und die Folgen somit auch unterschiedlich lang sein können.

Allerdings müssen für jedes Paar von Zuordnung $e_i \leftrightarrow \tilde{e}_{\tilde{i}}$ und $e_j \leftrightarrow \tilde{e}_{\tilde{j}}$ die folgenden Implikationen einhalten bleiben:

$$i < j \rightarrow \tilde{i} < \tilde{j} \tag{3.11}$$

$$i > j \rightarrow \tilde{i} > \tilde{j} \tag{3.12}$$

Dies bedeutet, dass für jedes Paar von Zuordnungen entweder eine Vorgänger- oder eine Nachfolgerbeziehung vorliegen muss.

Unter der Verwendung von Ähnlichkeiten der Elemente einer Folge und unter der Einhaltung der Implikationen kann durch eine Anordnung von zwei Folgen eine Zuordnung aller Elemente dieser Folgen bestimmt werden. Ein solches Verfahren ist aus der Bioinformatik bekannt, in der Segmente von genetischen Sequenzen von Nuclein-Säuren miteinander verglichen werden. Um eine solche Zuordnung zu finden, werden dafür Algorithmen, wie der Needleman-Wunsch Algorithmus [NW70] eingesetzt. Dieser Algorithmus wurde für die Arbeit modifiziert, um eine Zuordnung von Modellelemente vorzunehmen.[17]

Dazu ist es zunächst notwendig, die Ähnlichkeiten der Modellelemente in einer Substitutionsmatrix $S((e_i)_{i=1,\ldots,n}, (\tilde{e}_{\tilde{i}})_{\tilde{i}=1,\ldots,\tilde{n}})$. zusammenzufassen. Diese Matrix hat für jedes Paar von Elementen einen Substitutionswert $s(e_i, \tilde{e}_{\tilde{i}})$. Dieser ist positiv, wenn es eine hohe Ähnlichkeit zwischen den beiden Elementen gibt und er ist negativ, wenn es nur eine geringe Übereinstimmung gibt. Die in der vorliegenden Arbeit verwendete Bestimmung der Substitutionswerte ist in Formel 3.13 angeben:

$$h(i, \tilde{i}) = \begin{cases} \textbf{if } f_{\ddot{A}hnlichkeit}(e, \tilde{e}) \geq \Theta_{lokal} \textbf{ then } k_{sw} \cdot f_{\ddot{A}hnlichkeit}(e, \tilde{e}) \\ \textbf{else then } -k_{sw} \end{cases} \tag{3.13}$$

Mit Θ_{lokal} ist ein Schwellwert gegeben, der eine Zuordnung von niedrigen Ähnlichkeiten verhindert indem für die Ähnlichkeit unter dem Schwellwert ein negativer Wert angenommen wird. Dies bewirkt, dass Elemente mit niedriger Ähnlichkeit nicht zugeordnet werden. Eine solche fehlende Zuweisung ist bei der Evolution essentiell, da diese Modellelemente dann bei der Evolution hinzugefügt worden sind. Die Höhe des Substitionswertes kann mit dem Faktor k_{sw} konfiguriert werden, der als positiver Wert die Ähnlichkeit multipliziert und als Strafe bei Ähnlichkeiten unter dem Schwellwert fungiert. Durch den Faktor und den Schwellwert kann verhindert werden, dass eine längere Folge mehrerer geringer Ähnlichkeiten eine kürzere Folge mit hoher Ähnlichkeit aufwiegt. Dabei wird im ursprünglichen Needleman-Wunsch-Algorithmus bereits ein solcher Substitutionswert verwendet, allerdings dort konstant gehalten. Für die Zuordnung von Modellelementen bei der Evolution wird der Wert allerdings mit der Ähnlichkeit gesteigert, da nicht nur 100-prozentige Ähnlichkeiten berücksichtigt werden sollen. Dies trägt dazu bei, dass die Folgen möglichst intakt gehalten werden.

[17] Der Algorithmus wurde in seiner Grundversion im Rahmen der Forschungstätigkeit zusammen mit Abhishek Chakraborty entwickelt. Er wurde im Rahmen dieser Arbeit für die Evolutionsschritte verallgemeinert und verbessert sowie mit einem weiterer nachgelagerten Verfahren ergänzt.

Verfahrensvorschrift 3: Erstellung einer Anordnung von Modellelementfolgen

Eingabe: Zwei Folgen von Modellelementen $(e_i)_{i=1,\ldots,n}$ und $(\tilde{e}_{\tilde{i}})_{\tilde{i}=1,\ldots,\tilde{n}}$
Eingabe: Substitutionsmatrix $S((e_i)_{i=1,\ldots,n}, (\tilde{e}_{\tilde{i}})_{\tilde{i}=1,\ldots,\tilde{n}})$
mit Substitutionswerten $s(e_i, \tilde{e}_{\tilde{i}})$
Ausgabe : Anordnung der Folgen *anordnung*

$l =$ Strafe für Lücke
$lv =$ Strafe für Lückenverlängerung
/* Initialisierung der Wertungsmatrix für keinerlei Zuordnungen */
$w(i,0) := (-l-(i-1)\cdot lv)\ \forall\ i\ \in\ \mathbb{N}[1,n]$
$w(0,\tilde{i}) := (-l-(\tilde{i}-1)\ \cdot lv) \forall\ \tilde{i}\ \in\ \mathbb{N}[1,\tilde{n}]$
/* Schleife zum Finden der Zuordnung */
for $i=0;\ i<n;\ i=i+1$ **do**
  for $\tilde{i}=0;\ \tilde{i}<\tilde{n};\ \tilde{i}=\tilde{i}+1$ **do**
    $zuordnen = w(i-1,\tilde{i}-1) + s(i,\tilde{i})$
    $lücke_i = \begin{cases} \textbf{if } \text{startet eine Lücke } \textbf{then } w(i-1,\tilde{i}) - l, \\ \textbf{else then } w(i-1,\tilde{i}) - lv, \end{cases}$
    $lücke_{\tilde{i}} = \begin{cases} \textbf{if } \text{startet eine Lücke } \textbf{then } w(i,\tilde{i}-1) - l, \\ \textbf{else then } w(i,\tilde{i}-1) - lv, \end{cases}$
    $w(i,\tilde{i}) = \textbf{max} \begin{cases} zuordnen, \\ lücke_i, \\ lücke_{\tilde{i}}, \end{cases}$
    $array[i][\tilde{i}] =$Index $(i,\tilde{i})$ des Maximums
/* Rückverfolgung der Zuordnung über array[ ][ ] */
$i = n$ **and** $\tilde{i} = \tilde{n}$
while $(i,\tilde{i}) \neq (0,0)$ **do**
  if $array[i][\tilde{i}] = (i-1,\tilde{i}-1)$ **then**
    füge $\begin{Bmatrix} e_{i-1} \\ \tilde{e}_{\tilde{i}-1} \end{Bmatrix}$ in *anordnung* ein
  else if $array[i][\tilde{i}] = (i-1,\tilde{i})$ **then**
    füge $\begin{Bmatrix} e_{i-1} \\ - \end{Bmatrix}$ in *anordnung* ein
  else
    füge $\begin{Bmatrix} - \\ \tilde{e}_{\tilde{i}-1} \end{Bmatrix}$ in *anordnung* ein
  $(i,\tilde{i}) = array[i][\tilde{i}]$
Rückgabe *anordnung*

Die Verfahrensvorschrift 3 verwendetet eine Substitutionsmatrix, um eine Zuordnung der Folgen $(e_i)_{i=1,\ldots,n}$ und $(\tilde{e}_{\tilde{i}})_{\tilde{i}=1,\ldots,\tilde{n}}$ zu finden. Die Wertungen basiert zum einen auf den aufaddierten Substitutionswerten aus Formel 3.13 und zwei Strafwertungen für Lücken in der Anordnung. Hier wird eine Erweiterung des Needleman-Wunsch Algorithmus verwendet, die kleinere Lücken durch den Strafwert l härter bestraft als eine Verlängerung einer vorhandenen Lücke durch den geringeren Strafwert lv. Für die Zuordnung von Folgen in evolvierenden Modellen sollte deshalb immer $l > lv$ gelten. Diese Vorgabe hat den Grund, dass längere

aufeinanderfolgende Zuordnungen generiert werden sollen, sofern zumindest eine Ähnlichkeit über dem Schwellwert Θ_{lokal} existiert. Denn in diesem Fall scheinen die Transitions- bzw. Aufruffolgen zumindest eine grundsätzliche Ähnlichkeit aufzuweisen, was bei der Evolution grundsätzlich als modifiziertes Verhalten und nicht als Entfernen und Hinzufügen von neuem Verhalten erkannt werden sollte.

Die Verfahrensvorschrift bestimmt eine Wertung für mögliche Anordnungen der zwei Folgen. Dazu werden die zwei Folgen in einer Matrix angeordnet, die dann über die Indexe mögliche Anordnungen abgebildet. Entsprechend wird durch die Verfahrensvorschrift iterativ eine Wertungsmatrix gebildet indem in einer Schleife jeweils drei möglichen Fälle für jeden Index $(i, \tilde{i})$ der Wertungsmatrix betrachtet werden. Für die Elemente e_i und $\tilde{e}_{\tilde{i}}$ sind dies folgende Fälle:

1. e_i wird $\tilde{e}_{\tilde{i}}$ zugeordnet: In diesem Fall werden die Elemente in der betrachteten Anordnung zugeordnet. Entsprechend wird der Substitutionswert, der auf der Ähnlichkeit basiert, der bereits für diese Anordnung vorhandenen Wertung der vorherigen Iteration der Schleife hinzugefügt.
2. e_i wird nicht zugeordnet: In diesem Fall wird ein Element der Folge $(e_i)_{i=1,\ldots,n}$ nicht zugeordnet. Entsprechend wird eine Strafe auf die Wertung diese Anordnung addiert. Diese hängt davon ab, ob in der vorherigen Zuordnung der betrachteten Anordnung bereits eine Lücke besteht. Entsprechend wird lv oder l als Strafwert verwendet.
3. $\tilde{e}_{\tilde{i}}$ wird nicht zugeordnet: Dieser Fall ist äquivalent zum vorherigen Fall, nur für die Folge $(\tilde{e}_{\tilde{i}})_{\tilde{i}=1,\ldots,\tilde{n}}$.

Es wird für jeden Index der Wertungsmatrix immer die höchste Wertung der Fälle, also die beste Anordnung ausgewählt und als neue Wertung an diesem Index vermerkt.

Um am Ende die optimale Lösung wieder rekonstruieren zu können, wird zusätzlich in einer $array$-Variablen der Index der jeweils gewählten Lösung gespeichert. Dazu wird ausgehend von der Wertung $w(n, \tilde{n})$, der die Wertung der besten Anordnung enthält, immer der beste der oben erläuterten Fälle zurückverfolgt. Je nachdem, ob eine Lücke oder eine Zuordnung optimal ist, wird diese Zuordnung oder Lücke der optimalen Anordnung zugefügt bis beide Folgen vollständig zueinander angeordnet sind. Diese Anordnung ist dann unter den gegebenen Konfigurationen optimal.

Die Verfahrensvorschrift 3 beschreibt somit eine Funktion, welche für zwei Folgen eine optimale Anordnung berechnet. Wie diese Anordnung verwendetet wird ist modellabhängig und wird deshalb für die CPPS- und CBCPS-Modelltypen anschließend getrennt voneinander vorgestellt.

CPPS-Modelle:

Bei den betrachteten CPPS-Modellen handelt es sich um *signalbasierte Modelle*. Dabei sind die Transitionen, die Ereignisse von Signalen im Modell repräsentieren, als semantisch bedeutende Modellelemente für die Ähnlichkeit entscheidend.

Dabei wird in Maschinenzustandsmodellen ihrer Zuordnung zu Flanken und in Materialflussmodellen zur Konfiguration des Werkstückmaterials zusätzlich berücksichtigt. Nachfolgend werden für Evolutionsschritte allerdings nur Materialflussmodelle betrachtet, die zur weiteren Betrachtung der Evolutionsunterstützung in dieser Arbeit ausgewählt wurden.[18]

Nachfolgend wird die Zuordnung von Transitionen erläutert. Dadurch wird erreicht, dass für zwei Versionen das als gleich anzusehende Verhalten des CPPSs über die Transitionen der Modelle identifiziert werden. Um dies zu erreichen, wird nicht der Name, sondern nur der Kontext der Transitionen für die Ähnlichkeitsbestimmung verwendet. Dies hat den Grund, dass auch Varianten von CPPSen betrachtet werden sollen, deren Namenskonvention verschieden sind. Die Kontexte von Transitionen sind der Zeitkontext, der Orts- und der Typkontext. Für jeden dieser Kontexte wurde eine Ähnlichkeitsmaß definiert:

- **Zeitkontext:** Der Zeitkontext ist interessant, wenn betrachtet werden soll, wie eine Funktionalität im CPPS ausgeführt wird. Eine Transition hat ein Tupel des Zeitverhaltens $z_{k_l} = (\mu, \sigma, n, min, max)$ für jede seiner $k = k_1, \cdots, k_l, \cdots, k_m$ Konfigurationen. Wenn nur in einer der zwei zu vergleichenden Transitionen ein Tupel für die betrachtete Konfiguration vorliegt, ist die Ähnlichkeit dieser Konfiguration 0. Die Ähnlichkeit zweier Tupel von derselben Konfiguration muss über deren Abweichung zueinander bestimmt werden. Dies kann durch einen Zweistichproben-t-Test vorgenommen werden. Dieser ermöglicht mit den Mittelwerten zweier Stichproben[19] herauszufinden, ob diese aus einer gleichen Grundgesamtheit, also dem gleichen beobachteten Verhalten, stammen. Da im Fall des Zeitkontextes keine Ja/Nein-Entscheidung über die Gleichheit, sondern eine Ähnlichkeit bestimmt werden soll, wird dieses auf ein Intervall des Standardfehlers der Mittelwertdifferenzen definiert. Dazu wird, vergleichbar zum Zweistichproben-t-Test, dieser Standardfehler, wie in Formel 3.14 angegeben, bestimmt:

$$t_{z_{k_l} \leftrightarrow \tilde{z}_{k_l}} = \left| \sqrt{\frac{n \cdot \tilde{n}}{n + \tilde{n}}} \cdot \frac{\mu - \tilde{\mu}}{\sqrt{\frac{(n-1)\sigma^2 + (\tilde{n}-1)\tilde{\sigma}^2}{n + \tilde{n} - 2}}} \right| \tag{3.14}$$

  Dabei dient der Standardfehler als Maß für die Ähnlichkeit. Dieses ist entsprechend Formel 3.15 für ein Mindestwert $\Theta_{minZeit}$ des Standardfehlers definiert. Die Ähnlichkeit entspricht dann der Ähnlichkeit normiert auf das Intervall $[0, 1]$.

  An dieser Stelle kann keine Signifikanz des Nullhypothesentests erreicht werden, was aber auch nicht vorausgesetzt wird. Denn über das Intervall kann auch im Fall einer fehlenden Signifikanz mit dem Standardfehler der Mittelwertdifferenz ein geeignetes und in diesem Fall bestmögliches Ähnlichkeitsmaß definiert werden.

[18] Maschinenzustandsmodelle könnten vergleichbar dazu verwendet werden. Dies wurde in der vorliegenden Arbeit allerdings nicht weiter untersucht.

[19] Voraussetzung ist, dass die Stichproblem normalverteilt sind oder der zentrale Grenzwertsatz anwendbar ist, sowie dass die Standardabweichung (statistisch) gleich sind.

Mit der Ähnlichkeit für einzelne Tupel von Konfigurationen lässt sich dann die Ähnlichkeit des Zeitverhaltens von Transitionen in Formel 3.16 als Mittelwert aller ihrer Konfigurationen definieren.

$$ähnlichkeit^{Zeitkontext}_{z_{k_l} \leftrightarrow \tilde{z}_{k_l}} = \begin{cases} \textbf{if } t_{z_{k_l} \leftrightarrow \tilde{z}_{k_l}} > \Theta_{minZeit} \textbf{ then } 0 \\ \textbf{else then } 1 - (\frac{t_{z_{k_l} \leftrightarrow \tilde{z}_{k_l}}}{\Theta_{minZeit}}) \end{cases} \tag{3.15}$$

$$ähnlichkeit^{Zeitkontext}_{t_i \leftrightarrow \tilde{t_{\tilde{i}}}} = \frac{\sum_{l=0}^{m} ähnlichkeit^{Zeitkontext}_{z_{k_l} \leftrightarrow \tilde{z}_{k_l}}}{m} \tag{3.16}$$

- **Ortskontext:** Ein weiteres Kriterium der Ähnlichkeit ist der Ort, an dem die Ereignisse des an der Transition annotierten Signals beobachtet werden. Denn in CPPSen sollten Ereignisse, die an unterschiedlichen Ressourcen auftreten, eine geringere Ähnlichkeit aufweisen. Da Transitionen auch kombinierte Signale in Materialflussmodellen abbilden, berücksichtigt die Ähnlichkeitsfunktion alle im kombinierten Signal zusammengefassten Signale gleichgewichtet. Eine gängige Möglichkeit, die Ähnlichkeit von Mengen zu bestimmen ist der *Jaccard*-Koefizient [DD06], der die Größe der Schnittmenge zweier Mengen durch die Größe ihrer Vereinigungsmenge teilt. Da es sich bei Signalen um eine Multimenge handelt, wird die generalisierten Jaccard-Ähnlichkeit auf Vektoren mit der Anzahl der jeweiligen Ausprägungen angewendet. Der Ortskontext kann dabei direkt über die Ähnlichkeit des Modellattributtpaares $(ort = ausprägung)$ bestimmt werden, dass die im Meta-Model gezeigten Ausprägung der Enumeration $SOrt$ haben kann. Für jede Ausprägung wird die Anzahl der Signale gezählt und in einem Vektor $x = \{x_{magazin}, x_{crane}, x_{stamp}, x_{conveyor}\}$ für die Transition t_i respektive $\tilde{x} = \{\tilde{x}_{magazin}, \tilde{x}_{crane}, \tilde{x}_{stamp}, \tilde{x}_{conveyor}\}$ für die Transition $\tilde{t_{\tilde{i}}}$ festgehalten. Dann lässt sich mit der generalisierte Jaccard-Ähnlichkeit aus Formel 3.17 die Ähnlichkeit berechnen.

$$ähnlichkeit^{Ortskontext}_{t_i \leftrightarrow \tilde{t_{\tilde{i}}}} = \frac{\sum_{i \in SOrt} min(x_i, \tilde{x}_i)}{\sum_{i \in SOrt} max(x_i, \tilde{x}_i)} \tag{3.17}$$

- **Typkontext:** Zugeordnete Transitionen sollten für eine ähnliche Funktionalität im CPPS stehen, weshalb auch die Ähnlichkeit des Typkontextes für Transitionen betrachtet wird. Dabei gibt es bei der Beschreibung von Funktionalitäten Überlappungen. Für die verwendeten Kontexte dieser Arbeit (siehe [LHFL14b, LFL16]) sind diese Überschneidungen auszugsweise in Tabelle 3.3 angegeben. Die Tabelle zeigt die Zuordnung vom Modellattribut typ nach der Enumeration der Meta-Modelle für CPPS-Modelle als Relationswert $(typ_i \rightarrow \tilde{typ_{\tilde{j}}}) = r_{i\tilde{j}} \in [1, 0]$. Beispielsweise kann die vertikale Position $(S.XPosition)$ einer Komponente in einer ähnlichen Komponente eine horizontale Position $(S.YPosition)$ darstellen, weshalb diese eine Relation von 0,75 in der Tabelle aufweisen. Gleiche Ausprägungen haben eine Relation von 1.

  Unter diesen Relationen kann die Ähnlichkeit von zwei Mengen $Typ = \{typ_1, \ldots, typ_i, \ldots, typ_n\}$ und $\tilde{Typ} = \{\tilde{typ_1}, \ldots, \tilde{typ_{\tilde{j}}}, \ldots, \tilde{typ_{\tilde{n}}}\}$ durch Lösen eines linearen Zuordnungsproblems bestimmt werden. Mit den zu bestimmenden Entscheidungsvariablen

$$x_{i,\tilde{j}} = \begin{cases} \textbf{if } (typ_i \rightarrow \tilde{typ_{\tilde{j}}}) \; ist \; Teil \; der \; Lösung \textbf{ then } 1 \\ \textbf{else then } 0 \end{cases} \tag{3.18}$$

ergibt sich unter dem Nutzen der Relationswerte $r_{i\tilde{j}}$ von $(typ_i \rightarrow \tilde{typ_{\tilde{i}}})$ dabei das lineare Zuordnungsproblem:

$$\begin{aligned} &Z = max\ \textstyle\sum_{i=0}^{n} \sum_{\tilde{i}=0}^{\tilde{n}}\ r_{i\tilde{j}} \cdot x_{i,\tilde{j}} \\ &unter\ den\ Nebenbedingungen \\ &\textstyle\sum_{\tilde{j}=0}^{\tilde{n}} x_{i,\tilde{j}} \leq 1\ f\ddot{u}r\ i = 1, \ldots, n \\ &\textstyle\sum_{i=0}^{n} x_{i,\tilde{j}} \leq 1\ f\ddot{u}r\ \tilde{j} = 1, \ldots, \tilde{n} \end{aligned} \tag{3.19}$$

Aus der Lösung des Zuordnungsproblems ergibt sich dann vergleichbar zum Jacaard-Koeffizienten die Ähnlichkeit des Typkontextes:

$$\ddot{a}hnlichkeit^{Typkontext}_{t_i \leftrightarrow \tilde{t_{\tilde{i}}}} = \frac{2Z}{n + \tilde{n}} \tag{3.20}$$

Die gefundene Lösung gilt unter der Annahme, dass als Ähnlichkeit diejenige Kombinationen von Signalen aus der Menge der beiden kombinierten Signal gewählt wird, die die maximale Relation zueinander aufweist ohne ein Signal in der Kombination zweimal zu verwenden. Für nicht kombinierte Signale entspricht dieses Vorgehen dem Relationswert $r_{i\tilde{j}}$ der Typen der zwei betrachteten Signale.

Mit den Ähnlichkeiten der Kontexte und unter einer Gewichtung von Kontexten ($\omega_{Zeit},\ \omega_{Typ},\ \omega_{Ort} \in \mathbb{Q}_+$) kann die Ähnlichkeitsfunktion als gewichteter Mittelwert der zwei Transitionen berechnet werden. Dadurch können, wenn die Gewichte als 0 gesetzt werden, auch spezifische Kontexte bei der Ähnlichkeit ignoriert werden:

$$f_{t_i \leftrightarrow \tilde{t_{\tilde{i}}}} = \frac{\begin{aligned}\omega_{Zeit} \cdot \ddot{a}hnlichkeit^{Zeitkontext}_{t_i \leftrightarrow \tilde{t_{\tilde{i}}}} + \omega_{Ort} \cdot \ddot{a}hnlichkeit^{Ortskontext}_{t_i \leftrightarrow \tilde{t_{\tilde{i}}}} \\ + \omega_{Typ} \cdot \ddot{a}hnlichkeit^{Typkontext}_{t_i \leftrightarrow \tilde{t_{\tilde{i}}}}\end{aligned}}{\omega_{Zeit} + \omega_{Typ} + \omega_{Ort}} \tag{3.21}$$

Mit dieser Ähnlichkeit von Transitionen wird die Substitutionsmatrix gebildet und die Verfahrensvorschrift 3 für jede vergleichbare Folgen von Transitionen des Materialflussmodells ausgeführt.

Folgen sind in Materialflussmodellen vergleichbar, wenn sie eine vollständige Werkstoffinstanz im Sinne des Generierungsverfahrens sind, die Werkstoffe des gleichen Materials betrachten. Dabei entspricht eine vergleichbare Folge also einem Produktionspfad durch das betrachtete CPPS. Diese Produktionspfade können auch nachträglich durch eine Tiefensuche

Tabelle 3.3: Ausschnitt der Relationen der Ausprägungen des Typkontextes für CPPS-Modelle

	A.Move	**S.XPosition**	**S.YPosition**	**S.WPIdentify**	**S.WPDetect**	**A.WPHold**	**A.WPModify**
A.Move	1	0	0	0	0	0	0
S.XPosition	0	0,75	1	0	0	0	0
S.YPosition	0	1	0,75	0	0	0	0
S.WPIdentify	0	0	0	1	0,75	0,25	0,25
S.WPDetect	0	0	0	0,75	1	0,5	0,5
A.WPHold	0	0	0	0,25	0,5	1	0,25
A.WPModify	0	0	0	0,25	0,5	0,25	1

(siehe [TW15]) auf dem erzeugten Transitionsnetz des Materialflussmodells ermittelt werden indem ausgehend von den Transitionen ohne eine Vorstelle ein Baum aufgespannt wird und alle gefundenen Pfade der Tiefensuche gespeichert werden. Dabei werden mögliche Pfade bei der Tiefensuche durch die Modellelemente definiert. Denn alle Transitionen einer vergleichbaren Folge haben ein assoziiertes Element des Typs *Zeitverhalten*, dass ein Konfigurationselement des betrachteten Werkstoffmaterials besitzt.

Wenn für alle vergleichbaren Folgen eine Anordnung erzeugt wurde, muss eine globale, eindeutige Lösung für die Modellzuordnung gefunden werden. Dafür wird eine $|T| \times |\tilde{T}|$ Zuordnungsmatrix Z erzeugt, wobei T und $\tilde{T}$, die Menge aller Transition der beiden betrachteten Modelle sind. Die Werte $z(i, \tilde{i})$ entsprechen der summierten Anzahl der Zuordnungen $(t_i \rightarrow \tilde{t_{\tilde{i}}})$ in allen Anordnungen. Auf Basis dieser Matrix und den Entscheidungsvariablen

$$x_{i,\tilde{j}} = \begin{cases} \textbf{if } (t_i \rightarrow \tilde{t_{\tilde{j}}}) \; ist \; Teil \; der \; Lösung \; \textbf{then } 1 \\ \textbf{else then } 0 \end{cases} \tag{3.22}$$

kann dann, vergleichbar zur Zuordnung des Typkontextes, eine Lösung mit dem folgenden linearen Zuordnungsproblem gefunden werden:

$$\begin{aligned} &Z = max \; \textstyle\sum_{i=0}^{n} \sum_{\tilde{i}=0}^{\tilde{n}} \; z_{i\tilde{j}} \cdot x_{i,\tilde{j}} \\ &\\ &unter \; den \; Nebenbedingungen \\ &\textstyle\sum_{\tilde{j}=0}^{\tilde{n}} x_{i,\tilde{j}} \leq 1 \; für \; i = 1, \ldots, n \\ &\textstyle\sum_{i=0}^{n} x_{i,\tilde{j}} \leq 1 \; für \; \tilde{j} = 1, \ldots, \tilde{n} \end{aligned} \tag{3.23}$$

Die Problemdefinition basiert dabei auf der Annahme, dass möglichst viele Zuordnungen der Anordnungen von einzelnen Folgen erhalten bleiben.

Wenn mehr als eine optimale Lösung vorhanden ist, wird die optimale Lösung nach den folgenden Konfliktauflösungsregeln gewählt:

1. Die unterschiedlich gewählten Zuordnungen der optimalen Lösungen befinden sich in längeren Sequenzen von aufeinanderfolgenden Zuordnungen von Transitionen aller für diese Zuordnung betrachteten Anordnungen. Dies hat den Grund, dass, insbesondere lange Folgen von Transitionen, möglichst intakt gehalten werden sollen.
2. Es wird die Summe der Längen aller für diese Zuordnung betrachteten Anordnungen bestimmt, wobei die größere Summe ausschlaggebend ist. Dies hat den Grund, dass längere Folgen auf einen komplexen und damit bestimmenden Produktionsprozess hindeuten.
3. Die Lösungen werden als gleichwertig betrachtet und eine Lösung wird zufällig gewählt.

An dieser Stelle wären allerdings auch andere Konfliktauflösungen denkbar. Nach diesem Vorgehen kann eine optimale Zuordnung von Transitionen ermittelt werden.

Stellen, Verbindungen und das Zeitverhalten sind semantisch von den Transitionen abhängig. Deshalb wird deren Zuordnung über die Abhängigkeit zu Transitionen bestimmt. Bei Stellen wird dazu die vorgehende als auch nachgehende Transition berücksichtigt. Im Meta-Modell besteht dabei zu den Transitionen transitiv über die *StelleZuTranition*- und *TransitionZuStelle*-Elementen eine Referenz. Entsprechend werden diese Referenzen zur Zuordnung verwendet. Verbindungen werden über ihre Ausprägung und die referenzierten Transitionen und Stellen zugeordnet. Das Element des Zeitverhaltens wird ebenso durch ihre Referenz zu Transitionen zugeordnet, wobei nur Zeitverhalten mit gleicher Konfiguration einander zugeordnet werden.

Dadurch wird eine Zuordnung aller Modellelemente von zwei Materialflussmodellen bestimmt, die auf Ähnlichkeiten der einzelnen Transitionen untereinander beruht, die Folgen von Transitionen möglichst erhält und eine globale eindeutige Lösung darstellt.

CBCPS-Modelle:

Mit dem Repositorymodell wird die Struktur von Komponenten, Diensten und ihrer Methoden spezifiziert [RBB+], weshalb alle Methodenverhaltensmodelle auf dasselbe Repositorymodell zugreifen. Sie stellen somit Varianten dar, die auf der gleichen Systemstruktur agieren. Entsprechend ist eine Zuordnung auf Ebene des Repositorymodells nicht notwendig. Nichtsdestotrotz sind die Verbindungen zum Repositorymodell Kontextinformation der Methodenverhaltensmodellen, da sie z.B. beschreiben zu welcher Komponente und zu welchem Dienst die Methode gehört (Ortskontext). Entsprechend werden diese Verbindungen auch als Identifizierungsmerkmal bei der Zuordnung des Verhaltens von Methoden genutzt.

Dienstaufrufe sind das bedeutende semantische Element in Methodenverhaltensmodellen, weshalb *ExternerAufruf*-Elemente von besonderer Bedeutung sind. Nachfolgend wird gezeigt wie für CBCPSe die Zuordnung von mehreren Elementen des Typs *ExternerAufruf* bestimmt wird und, darauf basierend, über Assoziationen *InterneAktion*-Elemente zugeordnet werden.

Um Elemente des Typs *ExternerAufruf* miteinander zu vergleichen, wird der Kontext dieser Aktionen herangezogen. Orts- und Typkontext werden in Modellartefakten des Repositorymodells abgebildet, zu denen das *ExternerAufruf*-Element in Beziehung steht. Der Ortskontext wird durch den Dienst der aufgerufenen Methode beschrieben und der Typ durch die Elemente der Methodensignatur.

Für die Bestimmung der Ähnlichkeit dieser beteiligter Modellobjekte stehen im Meta-Modell nur die Namen der Modellelemente zur Verfügung. Eine Möglichkeit wäre, die Namen als persistente eindeutige Identifikatoren zu verwenden. Damit würde die Ähnlichkeit von Namen bei Gleichheit 1 betragen und bei Ungleichheit 0. Cohen et al. [CRF03] empfehlen allerdings für die Zuordnung von Enititätsnamen ein weniger stringentes Verfahren, wie die Jaro Ähnlichkeit [Win95] oder andere vergleichbare Ähnlichkeitsmaße (siehe dazu [DD06]). Die für CBCPS-Modelle verwendete Jaro-Ähnlichkeit $f_{jaro}(n, \tilde{n})$ für zwei beliebige Namen $n, \tilde{n}$ ist ein editierbasiertes Verfahren, das Mengen von gleichen Zeichen verwendet und als Resultat ein Ähnlichkeitsmaß im Intervall $[0, 1]$ zurückgibt.

Die Kontextähnlichkeiten werden unter diesem und weiteren Verfahren wie folgt bestimmt:

- **Zeitkontext:** Zuerst ist der Zeitkontext anhand der *BenötigterKontext*-Elemente der gesamten aufgerufenen Methode zu bestimmen und dann zu vergleichen. Dabei muss der Zeitkontext einer Methode mit der Zeit, die weitere externe Aufrufe dieser Methode benötigen, addiert werden. Für jede Methode ist der Zeitkontext durch die assoziierten *InterneAktion*-Elemente gegeben. Diese enthalten das gleiche Zeittupel $(\mu, \sigma, max, min, n)$, wie CPPS-Modelle. Der Zeitkontext wird dann iterativ bestimmt indem die Zeittupel der aufgerufenen Methode sowie, falls vorhanden, die Zeittupel der durch diese Methode verwendeten Methoden summiert werden. Die dazu verwendete iterative Verfahrensvorschrift 5 wird im Rahmen der Eigenschaftsbestimmung (Abschnitt 3.5.4) vorgestellt. Um die endgültige Ähnlichkeit der Modellelemente zu bestimmen wird, wie in CPPSen, der Standardfehler der Mittelwertdifferenz des berechneten Zeitkontextes verwendet, sodass der Zeitkontext für *ExternerAufruf*-Elemente analog zu den Formeln 3.14, 3.15 und 3.16 für Transitionen definiert ist.
- **Ortskontext:** Der Ortskontext lässt sich für ein *ExternerAufruf*-Element nur durch seinen Dienst identifizieren, da die Assoziationen zur Komponente in PCM nur einseitig modelliert sind. Somit kennt die Methode nicht die Komponenten, durch die sie im CBCPS angeboten oder benötigt wird (vergleiche Abbildung 3.16). Deshalb wird die Ähnlichkeit über den Dienstnamen des ersten Elements d und den des zweiten Elements $\tilde{d}$ bestimmt.

Für die Bestimmung der Ähnlichkeit des Ortskontextes wird dann die Jaro-Ähnlichkeit aus Formel 3.24 angewendet.

$$ähnlichkeit_{ea \to \tilde{ea}}^{ortskontext}(d, \tilde{d}) = \begin{cases} \textbf{if } m = 0 \textbf{ then } 0 \\ \textbf{else then } \frac{1}{3}\left(\frac{m}{|d|} + \frac{m}{|\tilde{d}|} + \frac{m - \frac{t}{2}}{m}\right) \end{cases} \tag{3.24}$$

Dabei beschreibt $|d|$ respektive $|\tilde{d}|$, die Länge der Namen in Zeichen. Mit m wird die Anzahl der Zeichen beschrieben, die in beiden Namen vorkommt aber den in Formel 3.25 gegeben Abstand bezüglich der Zeichenreihenfolge nicht überschreitet.

$$\frac{max(|d|, |\tilde{d}|)}{2} - 1 \tag{3.25}$$

Variable t gibt die Anzahl der notwendigen Vertauschungen (*transpositions*) an, die nötig sind, um unter Berücksichtigung von nur der in beiden Namen vorhandenen Zeichen m und unter Einhaltung ihrer ursprünglichen Reihenfolge eine identische Zeichenfolge für beide Namen zu erreichen.

- **Typkontext:** Für den Typkontext können aus der Methodensignatur der Methodenname n, sowie die Paramater $p = p_1, \cdots, p_i, \cdots, p_n$ und der Rückgabewert r mit ihren Datentypen verwendetet werden. Für die Methodennamen und den Namen des Rückgabewerts kann die Methode des Ortskontext analog angewendet werden. Parameter können in beliebiger Anzahl vorhanden sein. Für das dadurch entstehende Zuordnungsproblem muss für die Parameter $p = p_1, \cdots, p_i, \cdots, p_n$ eine $n \times n$ Matrix erstellt werden. Die Matrix wird mit der Anwendung der Jaro-Distanz $p_{i,\tilde{j}}$ (vergleiche Formel 3.24) für jedes Paar von Parametern und dem dazugehörigen Namen des Datentyps gefüllt. Analog zum Typkontext von CPPSen wird dann das Zuordnungsproblem für Parameter (vergleiche Formel 3.19) gelöst. Die Lösung ist die Ähnlichkeit der Parameter. Die Ähnlichkeit des Typkontextes ist dann definiert als Mittel zwischen den Ähnlichkeiten des Namens, der Parameter und des Rückgabetyps der Methode mit deren Gewichten ω_{Name}, $\omega_{Parameter}$ und $\omega_{Rückgabe}$ (vergleiche Formel 3.21).

Die abschließende Ähnlichkeitsfunktion für ein *ExternerAufruf*-Element ist Formel 3.21 folgend dann als gewichtetes Mittel der Kontextähnlichkeiten definiert.

Das Vorgehen zum Finden einer Anordnung ist identisch zu dem Vorgehen bei CPPS-Modellen, wobei Folgen von Elementen des Typs *ExternerAufruf* verwendet werden. Dadurch kann mit Verfahrensvorschrift 3 eine Anordnung der Folgen $(ea_i)_{i=1,\ldots,n}$ und $(\tilde{ea}_i)_{i=1,\ldots,n}$ gefunden werden. Wenn mehr als eine Zuordnung vorliegt, beispielsweise wenn das Modell eine Verzweigung aufweist, werden die entstehenden Konflikte analog aufgelöst.

Basierend auf dieser Zuordnung von *ExternerAufruf*-Elementen werden die weiteren Modellelemente des Methodenverhaltensmodells über ihre jeweiligen Abhängigkeiten zugeordnet. Dazu wird in dieser Arbeit eine Nachfolgerelation $\succ_N$ definiert, welche für jedes *InterneAktion*-Element angibt auf welches *ExternerAufruf*- oder *StartAktion*-Element es folgt. Da unter den Annahmen des Generierungsverfahren auf jedes *ExternerAufruf*-Element genau ein *InterneAktion*-Element folgt, ist dieses Vorgehen eindeutig. Alle weiteren Modellelement sind eindeutig von einem Element dieser Aktionstypen abhängig.

Durch das vorgestellte Vorgehen wird die gesuchte Zuordnung aller Modellelemente gefunden, die vorgibt welche Elemente durch eine Änderung neu hinzugefügt wurden.

Zusammenfassung der Zuordnung

In diesem Abschnitt wurde ein Verfahren vorgestellt, das es ermöglichen ähnliche Modellartefakte aus zwei Modellen zu bestimmen und durch diese eine Zuordnung von Modellelementen vorzunehmen. Für die Evolution ist dabei entscheidend, dass diese Ähnlichkeit eine Aussage darüber trifft, ob die Modellelemente die selbe Funktion in den zwei Modellen innehaben. Deshalb wurde ein allgemeines Verfahren über Anordnungen von Folgen definiert. Dieses verwendet semantisch bedeutende Elemente der Modelle und ihre Kontextinformationen, um die Ähnlichkeit der Elemente über ihre Positionen in den beiden Folgen und die Ähnlichkeit ihres Kontextes zu bestimmen. Das allgemeine Verfahren wurde mit unterschiedlichen Berechnungen für die Ähnlichkeiten auf CPPS- und CBCPS-Modelle angewendet.

Dabei funktioniert das Verfahren nur sinnvoll, wenn das Modell maßgeblich auf Folgen von Modellelementen basiert und die Position in der Folge eine semantische Bedeutung besitzt. Ansonsten ist aus dem vorgestellten Verfahren nur die Kontextähnlichkeiten der Modellelemente direkt zu verwenden. Des weiteren ist bei komplexen Modellen im vorgestellten Verfahren eine exponentielle Laufzeiterhöhung bezüglich der Anzahl der Folgen zu beobachten. Denn es werden grundsätzlich alle Folgen zu allen anderen Folgen verglichen. Dies müsste durch entsprechende Heuristiken für komplexe Modelle verbessert werden. Alternativ könnten auch allgemeine ähnlichkeitsbasierte Verfahren Anwendung finden, wobei diese nicht auf den Anwendungsfall der Evolutionsunterstützung zugeschnitten sind und deshalb vermutlich schlechtere Zuordnungen vornehmen.

Durch die bestimmte Zuordnung kann geschlussfolgert werden, dass ein nicht zugeordnetes Modellelement durch eine Änderung hinzugefügt bzw. entfernt wurde und dass ein zugeordnetes Modellelement in beiden Modellen vorhanden ist. Diese Information wird im folgenden Abschnitt genutzt, um eine Differenz und somit eine Reaktion des Evolutionsschrittes zu bestimmen.

3.5.3 Liften von Differenzen zur Wirkungsbestimmung

Um Änderungen zwischen verschieden Versionen oder Varianten eines Modellartefakts zu erkennen, wurde im Verhalten des Reaktionsartefakts (siehe Abbildung 3.19) bereits von Differenzen, sowie von Editierskripts gesprochen. Nachfolgend wird nun diese Reaktion im Sinne von Differenzen definiert.[20]

Durch die Differenzenberechnung mit semantischem Liften können einfache Differenzen mit primitiven Änderungen zweier Modelle m und $\tilde{m}$ durch Editierregeln in *Operationen* geliftet werden, um daraus ein **Editierskript** $m\delta\tilde{m}$ abzuleiten (siehe Abschnitt 2.2.5 und [Keh15]). Dies lässt sich durch folgende Funktion beschreiben:

$$f_{Differenz}(m, \tilde{m}, z(m, \tilde{m}), OP) = \delta(m, \tilde{m}) \tag{3.26}$$

Dabei ist bei Versionen m das Ursprungsmodell und $\tilde{m}$ das Folgemodell. In Varianten sind es jeweils parallel existierende Modelle. Weiterhin ist eine Menge von Operationen $OP = \{op_1, \cdot, op_i, \cdots, op_n\}$ gegeben, wobei eine **Operation** $op_i(LM) : RM$ angibt, wie ein Muster LM von Elementen in ein anderes Muster RM von Modellelementen transferiert werden kann.[21]

[20] Die Arbeit folgt grundsätzlich der Methodik von [Keh15, KKOS12, KKT11, KKT13], aber wendet diese auf Evolutionsschritte und die generierten Modelltypen an, weshalb eine abweichende Notation verwendet wird.

[21] In der vorliegenden Arbeit wird nachfolgend grundsätzlich nicht zwischen Operation und Regeln, wie im unterliegenden Ansatz, unterschieden.

Eine Anwendung der Operation ermöglicht unter einer bestimmten Modellzuordnung $z(m, \tilde{m})$ ein Fragment von m, das dem Muster LS entspricht, in ein Fragment von $\tilde{m}$, das dem Muster LS entspricht, zu transformieren. Dabei werden gleiche Elemente durch die Modellzuordnung vorgegeben. Durch diese **Operationsanwendungen** werden die primitiven Änderungen der Differenz in semantische, disjunkte Änderungsmengen $SA = \{pa_1, \cdots, pa_i, \cdots, pa_n\}$ partitionieren, wobei jede primitive Änderungen Teil genau einer Änderungsmenge ist. Eine primitive Änderung $pa_i = (atyp, e_1, \cdots, e_i, \cdots, e_n \cup \tilde{e}_1, \cdots, \tilde{e}_i, \cdots, \tilde{e}_n)\ mit\ e_i \in m,\ \tilde{e}_i \in \tilde{m}$ besteht aus einem Typ und einer beliebigen Menge von Elementen der betrachteten Modelle.

Zur Ableitung des Editierskript werden Abhängigkeiten der Operationen bestimmt und die konkreten Parameter der Modelle wiederhergestellt. Somit kann das Editierskript, wie in Formel 3.27 gezeigt, genutzt werden, um zu beschreiben, welche Operationen von primitiven Änderungen durchgeführt werden müssen, um m in $\tilde{m}$ zu transferieren. Dabei entspricht $m\delta\tilde{m}$ genau den Modellelementen, die die Differenz zwischen m und $\tilde{m}$ ausmachen.

$$m \rightarrow^{\delta(m,\tilde{m})} m \circ m\delta\tilde{m} = m \tag{3.27}$$

Die Hauptaufgabe bei der Übertragung der Methodik auf die Unterstützung von Evolutionsschritten ist die Spezifizierung der Operationen. Für diese Evolutionsunterstützung beschreibt eine Operation die zu erkennenden Änderungen im koevolvierten Modellartefakt. Die Beschreibung der Operation erfolgt durch einfache Graphoperationen des abstrakten Syntaxgraphen. Diese Arbeit verwendet für die Evolution drei dieser Graphoperationen, welche gleichzeitig auch die unterschiedenen Typen von Operationen in dieser Arbeit sind:

+ **Hinzufügen:** Ein Modellelement oder eine Referenz wurde in Modell $\tilde{m}$ im Vergleich zum Modell m hinzugefügt. Wenn zwei Versionen von Modellen betrachtet werden, ist dieses Element somit während des Evolutionsschrittes hinzugekommen.
− **Entfernen:** Dies ist der alternative Fall, in dem in Modell m ein Element oder eine Referenz vorhanden war, welche im Modell $\tilde{m}$ nicht mehr vorhanden ist.
○ **Modifizieren:** Die Elemente oder Referenzen sind sowohl im ersten als auch zweiten Modell vorhanden. Sie sind allerdings nicht identisch. Bei Modellelementen kann dies beispielsweise die Veränderung eines Attributs oder einer Ausprägung sein. Entgegen dem Hinzufügen oder Entfernen, wo keine Zuordnung der Elemente vorliegt, wurden die betrachteten Elemente bei einer Modifizierung zugeordnet.

Diese drei Änderungsarten und die daraus resultierenden Operationen werden nachfolgend jeweils für Materialfluss- und Methodenverhaltensmodelle getrennt voneinander vorgestellt.

CPPS-Modelle:

Für Materialflussmodelle sind zunächst die primitiven Änderungen zu betrachten. Primitive Änderungen werden für die drei Graphoperation *Einfügen*, *Entfernen* und *Modifizieren* aus dem Meta-Modell abgeleitet. Beispiele dieser primitiven Änderungen bezüglich eines Materialflussmodell sind folgend genannt:

1. $erzeugeElement(Transition)$
2. $entferneElement(Stelle)$
3. $erzeugeReference(zu, StelleZuTransition, Transition)$
4. $entferneReference(von, Transition, TransitionZuStelle)$
5. $modifiziereElement(Signal, \{ort, MAGAZIN")$

Die primitiven Änderungen beschreiben zum einen das Hinzufügen und Entfernen von Elementen, wie Transitionen, Stellen oder Verbindungen (Beispiele 1. und 2.). Ferner werden auch

Referenzen, wie beispielsweise hinzugefügte Assoziationen zwischen den Transitionen und ihrer Verbindungen abgebildet (Beispiele 3. und 4.). Eine modifizierende primitive Änderung wird durch die Veränderung der Attribute von Elementen vorgenommen, wie z.B. das Setzen des Attributes ort an einem Signalelement (Beispiel 5.). Eine Operation, wie auch eine einfache Differenz kann dabei eine große Anzahl dieser primitiven Änderungen enthalten.

Operation	Name	Typ	Modellelemente (erhalten / hinzu.[entf.])	Weitere Objekte (Referenzen / Attribute / Parameter)	Bedeutung für Folgemodell
A	Folgetransition	+[−]	2 / 4	16 / 0 / 6	Ein (kombiniertes) Ereignis wurde [nicht mehr] nach einer Sequenz beobachtet
B	Vortransition	+[−]	2 / 4	16 / 0 / 6	Ein (kombiniertes) Ereignis wurde [nicht mehr] vor einer Sequenz beobachtet
C	Einfügetransition	+[−]	3 / 4	20 / 0 / 7	Genau ein (kombiniertes) Ereignis wurde [nicht mehr] zwischen zwei Sequenzen beobachtet
D	Zwischentransition	+[−]	3 / 5	22 / 0 / 8	Ein (kombiniertes) Ereignis wurde [nicht mehr] zwischen zwei Sequenzen beobachtet
E	Fusionstransition	+[−]	3 / 3	14 / 0 / 6	Zwei vorher unverbundene/[verbundene] Sequenzen haben [nicht mehr] einen gemeinsamen Teil
F	Signal	+[−]	1 / 1	1 / 3 / 5	Ein (kombiniertes) Ereignis hat ein Signal mehr/[weniger]
G	Zeitverhalten	+[−]	1 / 2	2 / 7 / 10	Ein Ereignis zeigt [nicht mehr] eine bestimmte Werkstückausprägung
H	ZeitverhaltenStelle	+[−]	1 / 2	2 / 7 / 10	Eine Stelle zwischen zwei Ereignis zeigt [nicht mehr] eine bestimmte Werkstückausprägung
I	Ort	◯	1 / 0	0 / 1 / 2	Das modellierte Ereignis hat einen anderen Ortskontext
J	Typ	◯	1 / 0	0 / 1 / 2	Das modellierte Ereignis hat einen anderen Typkontext
K	Zeit	◯	1 / 0	0 / 5 / 6	Das Zeitverhalten des Ereignisses einer Werkstückausprägung hat sich verändert.

Tabelle 3.4: Operationsmenge von Materialflussmodellen

Deshalb werden zum besseren Verständnis der Änderungen manuelle Operationen für Materialflussmodelle definiert. Bezüglich der avisierten Evolutionsunterstützung beschreiben diese Operationen genau die Änderungen, welche durch die Generierungsverfahren bei der Koevolution des Modells entstehen können. Diese Modelle sind eine Untermenge aller durch das Meta-Modell möglichen Modelle. Allerdings können dadurch die Operationen die Evolution in einem gut definierten Rahmen von möglichen Änderungen beschreiben.

Tabelle 3.4 zeigt die Operationsmenge OP von Materialflussmodellen. Dabei wird neben dem Namen der Operation auch der Typ (*Hinzufügen* $+$, *Entfernen* $-$, *Modifizieren* $\bigcirc$) dieser Operation angegeben. In der nächsten Spalte wird die Anzahl von erhaltenden und hinzugefügten oder entfernten Modellelementen aufgeführt. Diese ermöglicht zusammen mit der Anzahl von hinzugefügten oder entfernten Assoziationen, Attributen und Parametern eine Einschätzung der Komplexität, der durch die Operation identifizierten Änderung.

Mit den Operationen A-E wird die Struktur von Transitionen im Materialfluss abgebildet. Die Struktur entspricht den durch das Generierungsverfahren zusammengeführten Ereignissequenzen. Dabei kann die generierte Struktur verschiedene Fälle bezüglich der Differenz von Transitionsfolgen hervorbringen. Zunächst kann am Anfang oder am Ende einer Ereignissequenz ein zusätzliches (kombiniertes) Ereignis beobachtet werden. Dies entsteht während der Evolution beispielsweise durch die Erweiterung oder Reduktion zum Start oder zum Ende des Produktionsprozesses (Operation A+B).

Für eine Erweiterung in der Mitte einer Ereignissequenz wurden vorhandene Verbindungen einer Transitionsfolge entfernt und nach der Einfügung der zusätzlichen Transitionen wieder an der gleichen Stelle miteinander verbunden. Eine Reduktion trennt die Transitionsfolge auf und fügt über entsprechende verbundene Referenzen die getrennte Folge wieder zusammen. Dies kann für nur eine Transition passieren, welche das Entfernen und Hinzufügen der Referenzen an der vorhandenen Folge in einer Operation beschreibt (Operation C), oder bei mehreren Transitionen durch eine Kombination von Operation D und E. Zwei Operationen sind hierbei notwendig, da es für den unterliegenden Mechanismus zum Suchen von Operationen in der Differenz problematisch ist, eine beliebige Anzahl von Modellelementen als Muster der Differenz zu finden. Beide dieser Fälle beschreiben beispielsweise eine Änderung, durch den der Produktionsprozess um zusätzliche Sensoren erweitert wird.

Operation E bildet das Zusammenführen vorher getrennte Ereignissequenzen zu einem nun gemeinsamen Teil ab, da das Generierungsverfahren z.B. den gemeinsamen Teil der Transitionen fusioniert hat. Dies ist genauso auch für den Fall einer *Entfernung* der Operation denkbar (Typ $-$in Abbildung 3.4). In diesem Fall haben Ereignissequenzen nun keinen gemeinsamen Teil mehr, da diese z.B. bei der Generierung nicht mehr fusioniert werden. Dabei sind die Operationen für das Einfügen von Transitionen in eine Folge die komplexesten, da sowohl Verbindungen entfernt als auch hinzugefügt werden müssen und bestimmte Voraussetzungen vorliegen müssen. Alle Operationen werden iterativ identifiziert, weshalb in allen Fällen zwei oder mehr Transitionen hinzugefügt oder entfernt werden können. Dazu werden die primitiven Änderungen sukzessive den entsprechenden semantischen Änderungsmengen der Operationen zugeteilt bis keine weiteren primitiven Änderungen mehr vorliegen.

Neben strukturellen Änderungen der Transitionen und Stellen, können auch Änderungen bei den annotierten Signalen der Transitionen auftreten. Operation F gibt an, dass ein zusätzliches Signal zu einer Transition hinzugefügt oder ein vorher vorhandenes Signal entfernt wurde. In diesem Fall wurde das Modellelement des Signals mit seinen Attributen hinzugefügt. Um dabei eine korrekte Erkennung dieser Änderung zu gewährleisten, ist es notwendig, dass die Transitionen beider Modelle zugeordnet wurden. Denn ansonsten würde das Entfernen und

Hinzufügen jeweils einer Transition mit den entsprechenden Signalen beobachtet werden, welches durch die auf das Ursprungsmodel angewendete Operationen allerdings auch zum gleichen Folgemodell führen würde. Dies würde allerdings einer anderen Aussage bezüglich der Evolution entsprechen, da in diesem Fall alle Signale entfernt und ersetzt werden würden und bei einer Änderung durch Operation F nur ein Signal zu den bestehenden Signalen hinzugefügt würde. Aus der Sichtweise des Nutzers würde das Hinzufügen und Entfernen bedeuten, dass er z.B. eine Funktionalität im CPPS entfernt hat und eine unabhängige neue Funktionalität hinzugefügt hat. Eine Modifizierung würde bedeuten, dass er die vorhandene Funktionalität angepasst hat.

Die Veränderungen bei verschiedenen Werkstücktypen werden durch die Operationen G und H abgebildet. Die Operationen identifizieren keine strukturelle Änderung des CPPSs sondern eine Änderung bei der Beobachtung von unterschiedlichen Werkstücktypen. Diese Änderungen treten auf, wenn im Folgemodell eine veränderte Werkstückausprägung wahrgenommen wird. Beispielsweise kann eine neue Werkstückausprägung an einem CPPS beobachtet werden oder diese nicht mehr beobachtet werden. Der Unterschied besteht darin, dass Operation G Änderung an Transitionen und Operation H Änderungen an Stellen des Modells identifiziert.

Neben Transitionen können auch die Kontextinformationen dieser gleich zugeordnet Elemente modifiziert werden. Daraus resultieren die Modifizierungsoperationen I, J und K. Operation I und J beschreiben Modifizierungen im Element des *Signals*. Da das Namensattribut in CPPS-Modellen unberücksichtigt bleiben soll, kann nur eine Modifizierung des Typs- und des Ortsattributs stattfinden. Eine Modifizierung des Orts bedeutet für die Evolution, dass ein Ereignis in einer anderen Komponente wahrgenommen (Operation I). Bei der Veränderung des Typattributs wird ein wahrgenommene Signalereignis im Folgemodell durch einen veränderten Sensor oder Aktor wahrgenommen wird (Operation J).

Die Modifizierung des Zeitkontextes wird durch das Element *Zeitverhalten* im Materialflussmodell beschrieben. Dies umfasst mehr als ein Attribut, da alle fünf Attribute des Zeittupels einer Modifizierung unterliegen können. Dies gilt insbesondere auch, wenn keine Änderung im Sinne des in Abschnitt 3.2 definierten Evolutionsverständnisses vorliegt. Denn das Zeitverhalten enthält auch Parameter der Beobachtung des CPPSs, wie der Anzahl der an der Transition beobachteten Werkstücke. Des Weiteren unterliegt das Zeitverhalten auch kleinen, nicht signifikanten Schwankungen der Messungen, die beispielsweise durch die Zykluszeit der Steuerung, die Abtastrate der Beobachtung oder anderen Hardware- und Softwareeffekten entstehen. Dadurch werden bezüglich der Attributänderungen primitive Änderungen durch eine Anwendung der modifizierenden Operation H erkannt. Um nicht relevanten Operationsanwendungen herauszufiltern, wird ein Nachverarbeitung von Operationsanwendungen des Zeitkontextes vorgenommen. Diese filtert die Anwendungen der Operation K bezüglich einer signifikanten Änderung des Mittelwertes μ mit Hilfe der Ähnlichkeitsbestimmung des Zeitkontextes in der Zuordnung von Modellelementen (siehe Abschnitt 3.5.2). Dazu wird ein Schwellwert für den berechneten t-Wert verwendet. Dieser Schwellwert ist allerdings stark von dem beobachteten CPPS abhängig. Nur wenn eine Signifikanz über den Schwellwert festgestellt wird, wird die Operationsanwendung in der Bestimmung der Reaktion von Evolutionsschritten betrachtet.

Diese Menge von Operationen stellt die in Formel 3.26 beschriebene Operationsmenge OP dar. Entsprechend kann mit den Transformationskonzepten des verwendeten Differenzenansatz (siehe Abschnitt 2.2.5) für zwei Modelle m und $\tilde{m}$ unter einer Zuordnung $z(m, \tilde{m})$ (siehe Abschnitt 3.5.2) ein Editierskript berechnet werden.

Dabei werden, neben der Abhängigkeitsanalyse, alle Objekt- und Wertparameter der Operationen wiederhergestellt. Da das Editierskript in der avisierten Evolutionsunterstützung für Varianten nicht nur auf das Ursprungsmodell m angewendet wird, sondern auch auf eine unabhängige dritte Variante $\hat{m}$ angewendet werden soll, muss für diese später erläuterte Evolutionsunterstützung eine Zuordnung $z(\hat{m}, m)$ bestimmt werden. Deshalb werden zusätzlich zum eigentlichen Editierskript die Kontextinformationen der Elementparameter des Editierskript transferiert. Dieser **Modellkontext** muss die Kontextinformationen aller Elementparameter und die Folgen, auf denen die Modellelemente der Elementparameter liegen, umfassen. Da es sich bei gesamten Folgen von Signalereignissen des Produktionsprozesses um kritische Geschäftsdaten handelt, ist in Abschnitt 4.6.2 der operativen Betrachtung dieser Arbeit die Verwendung von einfachen Schutzstrategien für diesen Transfer des Modellkontextes vorgesehen.

Durch das für CPPSe vorgestellte Verfahren kann die Verhaltensfunktion des Reaktionsartefakt erfüllt werden. Im artefaktbasierten Evolutionsprozess werden dabei das Editierskript und sein Modellkontext als Wirkungsobjekt über die Ausgangsschnittstelle i_1 transferiert (vergleiche Abbildung 3.18 und 3.19). Manuelle Kontexte der An-Dimension des CRI-Modells werden parallel über Ausgangsschnittstelle t_2 für die bestimmte Wirkung transferiert.

CBCPS-Modelle:

Bei der Betrachtung von CBCPS-Modellen stellt das Methodenverhaltensmodell jeder Methode genau eine Variante dar. Diese werden bezüglich ihrer Differenzen untersucht, wobei ein Editierskript aussagt, was in der Methode verändert werden muss, um von dem Modell einer Methode zum Modell einer anderen Methode zu gelangen. Dies ermöglicht es beispielsweise, ein operationsbasiertes Ähnlichkeitsmaß für Evolutionsschritte zu erstellen oder Synergieeffekte bei der modellgetriebenen Softwareentwicklung aufzudecken indem Änderungen zwischen Modellen propagiert werden können.

Dazu ist zunächst zu betrachten, welche primitiven Änderungen in den Assoziationen des Elements *MethodenVerhalten* möglich sind. Dies ist vergleichbar zu Materialflussmodellen und betrifft insbesondere das Hinzufügen und Entfernen von externen Aufrufen oder internen Aktionen. Weiterhin kann insbesondere der Zeitverbrauch modifiziert werden. Der Kontrollfluss dieser Elemente wird durch das Hinzufügen und Entfernen der Assoziationen *Vorgänger* und *Nachfolger* realisiert.

Als Operationsmenge OP ergeben sich dabei die in Tabelle 3.5 gezeigten Operationen. Das Minimalmodell von Methodenverhaltensmodellen unter dem Generierungsverfahren umfasst die Elemente *StartAktion*, *StopAktion* und *InterneAktion*. Dabei hat jedes erzeugte *InterneAktion*-Element auch bereits ein *BenötigterKontext*-Element für das Zeittupel. Von diesem Minimalmodell ausgehend erzeugt das Generierungsverfahren komplexere Kontrollflüsse. Da es einen festen Start und ein festes Ende der Folge gibt, werden nur *ExternerAufruf*- und *InterneAktion*-Elemente eingefügt. Dabei wird aufgrund des Generierungsverfahren immer ein Paar dieser Aktionen eingefügt. Es wird also ein Element des Typs *ExternerAufruf* und eines des Typs *InterneAktion* eingefügt. Eingefügte Aktionen bedürfen dabei immer des Entfernens der Nachfolge- und Vorgänger-Assoziation.

Für eine einzelnes Aktionspaar werden die zwei Elemente und ihre Referenzen hinzugefügt (Operation A). Wenn mehr als ein solches Aktionspaar eingefügt wird, bedarf es für die Erkennung, wie bei Materialflussmodellen, zweier Operationen (Operation B+C). Sollen zusätzliche Aktionspaare zwischen diesen beiden Operationen eingefügt werden, kann dies durch die Operation D für Folgeaktionen erkannt werden.

Operation	Name	Typ	Modellelemente (erhalten / hinzu.[entf.])	Weitere Objekte (Referenzen / Attribute / Parameter)	Bedeutung für Folgemodell
A	Einfügeaktionen	+[−]	4 / 3	12 / 2 / 9	Genau ein Aufruf wurde [nicht mehr] zwischen zwei Sequenzen beobachtet
B	Zwischenaktionen	+[−]	3 / 3	10 / 2 / 8	Ein Aufruf wurde [nicht mehr] zwischen zwei Sequenzen beobachtet
C	Fusionsaktionen	+[−]	4 / 3	10 / 2 / 9	Zwei vorher unverbundene/[verbundene] Sequenzen haben [nicht mehr] einen gemeinsamen Teil
D	Folgeaktionen	+[−]	3 / 3	8 / 2 / 8	Ein Aufruf wurde [nicht mehr] nach einer Sequenz beobachtet
E	Verzweigung	+[−]	3 / 5	15 / 5 / 13	Es wurden zwei Sequenzen mit [keinem] unterschiedlichem Teil [mehr] beobachtet
F	Stop	+[−]	2 / 1	3 / 0 / 3	Eine unterschiedliche Sequenz endet [nicht mehr]
G	Zeit	◯	1 / 0	0 / 1 / 2	Das Zeitverhalten der Methode hat sich verändert.
H	Signatur	◯	2 / 0	2 / 0 / 2	Der zugeordnete Aufruf nutzt eine andere Methode.
I	Wahrscheinlichkeit	◯	1 / 0	0 / 1 / 2	Die Wahrscheinlichkeiten einer Verzweigung haben sich geändert.

Tabelle 3.5: Operationsmenge von Methodenverhaltensmodelle

Dieser Fall tritt auch auf, wenn ein Element des Typs *VerzweigungsAktion* eingeführt wird (Operation E). Dieses ersetzt eine bestehende Referenz zwischen *InterneAktion*- und *ExternerAufruf*-Elementen durch eine Verzweigung des Kontrollflusses mit anschließenden zwei Elementen des Typs *InterneAktion* und den *BenötigterKontext*-Elementen. Dies ist die komplexeste erkannte Änderung, da durch die Operation fünf Elemente hinzugefügt werden, die mit zwei erhaltenden Elementen assoziiert sind. Zusätzlich wird das Attribut der Wahrscheinlichkeit in der Verzweigung gesetzt. Die Zusammenführung bedarf nur eines *InterneAktion*-Elements und des *VerzweigungsAktion*-Elements (Operation E). Wenn keine Zusammenführung stattfindet, kann das Generierungsverfahren das Verhalten der Methode auch durch ein zusätzliches

StopAktion-Element abgeschlossen werden (Operation F). Man beachte, dass ein zusätzliches *StartAktion*-Element dann nicht nötig ist.

Bezüglich einer Modifikation sind hier drei verschiedene Operationen möglich: Zum einen kann sich der Zeitkontext innerhalb des *BenötigterKontext*-Elements verändern. Da nur ein Zeitverhalten generiert wird, handelt es sich immer um eine Modifizierung des Attributswertes $bedarf$ (Operation G). Da das Repositorymodell beim Vergleich von Varianten in allen verglichenen Modellen identisch ist, besitzen die Signaturen keine Differenzen. Dennoch können sie die Referenzen der Elemente des Typs *ExternerAufruf* verändern sofern zwei unterschiedliche, aber ähnliche Methodenaufrufe zugeordnet werden (Operation H).

Die letzte Operation beschreibt eine Änderung des Attributs *Wahrscheinlichkeit*, das durch die Anzahl der beobachteten Aufrufe der Nachrichtensequenzen gebildet wird (Operation I). Dieses bedarf, wie auch das benötigte Verhalten, einer Nachverarbeitung, die nicht signifikante Wertveränderungen mit einem Schwellwert herausfiltert.

Die Operationen definieren die Operationsmenge OP des Differenzbildungsverfahrens für CBCPS-Modelle. Ein Editierskript wird entsprechend dem beschriebenen Vorgehen bei Materialflussmodellen gebildet. Der Modellkontext entspricht dabei dem Kontext der *ExternerAufruf*-Elemente, welche als Elementparameter in den Operationen verwendet werden. Dies umfasst insbesondere die assoziierte Signatur und den daraus abgeleiteten Orts- und Typkontext. Bei Elementparametern für Elemente des Typs *InterneAktion* wird wie bei der Zuordnung auf die Nachfolgerelation zurückgegriffen. Sofern der gesamte Modellkontext des Evolutionsschrittes vorliegt, sind dies alle Aufruffolgen, bei denen ein *ExternerAufruf*-Element betroffen ist. Der An-Kontext ist wie bei CPPS-Modellen durch den Nutzer im CRI-Modell spezifizierbar.

Zusammenfassung der Wirkungsbestimmung

Zusammenfassend kann über die Zuordnung von koevolvierenden Modelle und die Anwendung eines Differenzbildungsverfahren, die Wirkung eines Evolutionsschrittes modellbasiert beschreiben werden. Diese liegen als asymmetrische Differenz vor. Das heißt, dass dadurch spezifiziert ist welche Verhaltensänderungen von einer Modellversion zu der nächsten Modellversion vorgenommen wurden. Durch die Koevolution entspricht diese Reaktion auch der im CPS vorgenommen Änderung. Diese kann über das Editierskript zusammen mit dem Modellkontext als Teil eines Evolutionsschrittes im CPS propagiert werden und als Modelltransformation auf ähnliche Modelle angewendet werden. Zusammen mit dem expliziten Kontext wird es so möglich durchgeführte Änderungen automatisiert zu dokumentieren.

Die Qualität dieser Dokumentation hängt neben der schon diskutierten Beobachtung und Koevolution von einer sinnvollen Zuordnung und Operationsmenge ab. Eine schlechte Zuordnung würde bewirken, dass falsche Schlussfolgerungen über hinzugefügte oder verbleibende Modellelemente gezogen werden. Im schlechtesten Falle macht dies die Reaktion unbrauchbar. Die Operationsmenge bestimmt, wie vollständig und aussagekräftig die beschriebenen Operationen der Wirkung sind. Eine fehlerhafte Operationsmenge würde die Kompression der Anzahl von Änderungen bzw. Operationen vermindern, sodass im schlechtesten Fall weiterhin nur primitive Änderungen vorliegen, die durch den Nutzer nur schwer zu verstehen sind

3.5.4 Ergebnis auf Basis von Eigenschaftsveränderungen

Neben der Bestimmung, an welchen Stellen die Wirkung erzielt werden kann, sollte auch das erwartete Ergebnis der Anwendung eines Evolutionsschrittes beschrieben werden. Denn die Erhaltung und Verbesserung der Qualität, wie z.B. der Performanz eines CPSs, ist ein Hauptgrund

für Evolution [GRG+15]. Um eine Bewertung von Evolutionsschritten zu ermöglichen, werden dafür nichtfunktionale Eigenschaften als Bewertungsgrundlage in dieser Arbeit gewählt, da sie entgegen von funktionalen Eigenschaften einen allgemeinen Charakter für eine generelle Bewertung von Evolution haben [Lad18]. Dabei sind allerdings prinzipiell alle Eigenschaften für eine Verwendung geeignet, die aus den Modelltypen abgeleitet werden können.

Das Ergebnisartefakt besitzt drei Eingangskanäle, mit denen das Ergebnis bestimmt werden kann (siehe Abbildung 3.18). Dies sind die Versionen eines Modellartefakts über die Schnittstelle i_1^M, zusätzliche explizite Informationen des Nutzers über die Schnittstelle i_2^O, sowie Erfahrungen mit dem betrachteten Evolutionsschritt von anderen CPSen über die Schnittstelle i_3^S. Abbildung 3.21 beschreibt, wie die daraus resultierende Verhaltensfunktion $V^{Ergebnisartefakt} : \overrightarrow{\{i_1^M, i_2^O, i_3^S\}} \rightarrow \overrightarrow{\{o_1^E\}}$ mit der Ausgangsschnittstelle o_1^E des Effekts und seines Kontextes realisiert wird.

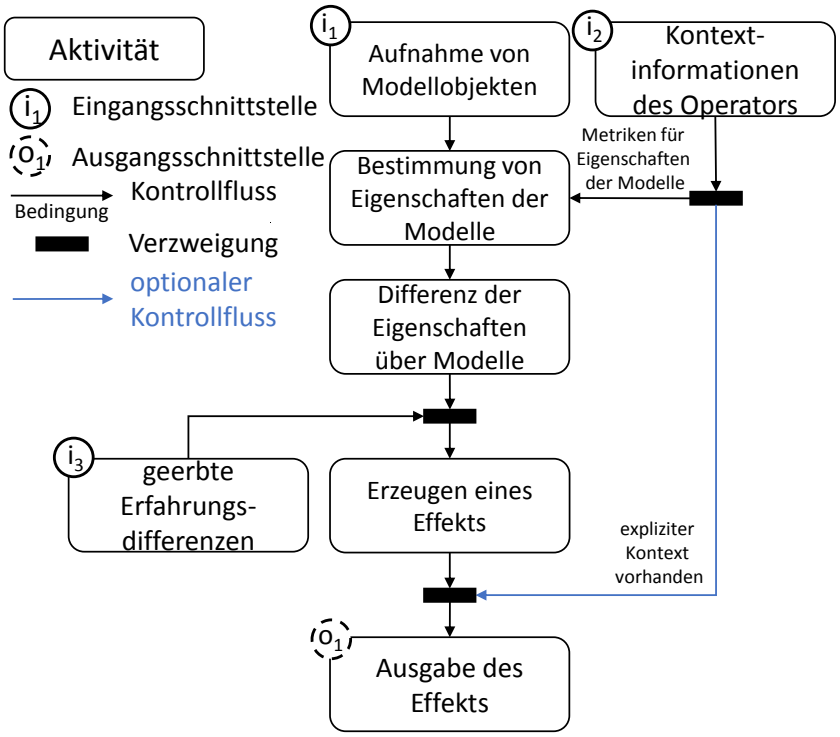


Abbildung 3.21: Artefaktverhalten zur Erstellung von Effekten als Ergebnis eines Evolutionsschrittes

Abbildung 3.21 zeigt das realisierte Verhalten des Ereignisartefakts. Zunächst ist es für die gegebenen Versionen der Modellartefakte notwendig, Eigenschaften zu berechnen, indem diese als konkrete, quantifizierte Kennzahlen ausgedrückt werden [Int05]. Für diesen Zweck muss eine Metrik definiert werden, die folgende Funktion zur Bestimmung des Werts einer *Kennzahl* $x_m^{es_i}$ für beliebige Eigenschaften $es_i \in \{es_1, \cdots, es_i, \cdots, e_n\}$ realisiert:

$$f_{Analyse}^{es_i}(m) = x_m^{es_i} \in \mathbb{Q}_+ \tag{3.28}$$

Die Metriken sowie die darüber bestimmten Eigenschaften sind modellabhängig und durch die Aussagekraft des Modells beschränkt. Eigenschaften beziehen sich in dieser Arbeit immer auf beobachtete technische CPSe und sind über deren Verhalten definiert. Welche Eigenschaften betrachtet werden, wird durch den Nutzer und seine für die Modelle bereitgestellten Metriken vorgegeben.

Nach der Berechnung von Kennzahlen für ein Ursprungsmodell vor einer Änderung m und eines Folgemodells nach einer Änderung $\tilde{m}$ kann das *Ergebnis dieser Änderung* berechnet werden. Mit diesen Kennzahlen wird die absolute und relative Differenz der betrachteten Änderung berechnet. Die absolute Differenz ist die Subtraktion der Kennzahl des Folgemodells von der Kennzahl des Ursprungsmodells und die relative Differenz ist die auf das Ursprungsmodell bezogene relative Erhöhung oder Verringerung des Wertes.

$$absDiff^{es_i}_{\tilde{m}-m} = f^{es_i}_{Analyse}(\tilde{m}) - f^{es_i}_{Analyse}(m) \tag{3.29}$$

$$relDiff^{es_i}_{\tilde{m}-m} = \begin{cases} \textbf{if } f^{es_i}_{Analyse}(m) = 0 \textbf{ then } \infty \\ \textbf{else then } \dfrac{absDiff^{es_i}_{\tilde{m}-m}}{f^{es_i}_{Analyse}(m)} \end{cases} \tag{3.30}$$

Da Evolutionsschritte unter verschiedenen CPSen ausgetauscht werden können, können zwei verschiedene Informationsquellen für die Eigenschaftsdifferenzen existiere. Zum einen ist dies die oben beschriebene Berechnung der Kennzahlen auf Basis des Eingangskanals von Modellartefakten ($c^M_{Modelle}$). Zum anderen können aber auch bereits vorhandene Ergebnisse von Evolutionsschritten genutzt werden, die zur Verfügung stehen, wenn ein Evolutionsschritt bereits im CPS oder durch ein anderes CPS durchgeführt wurde. Eine solche **Erfahrung** des vorhandenen Evolutionsprozesses wird dem Ergebnis über den Eingangskanals $c^M_{Modelle}$ beigefügt (vergleiche das Artefaktverhalten in Abbildung 3.18). Dieser Austausch von Erfahrungen wird bei der Realisierung eines Evolutionsprozesses und der wissenstragenden Komponente noch weitergehend erläutert (siehe Abschnitt 3.6.2 und Kapitel 4).

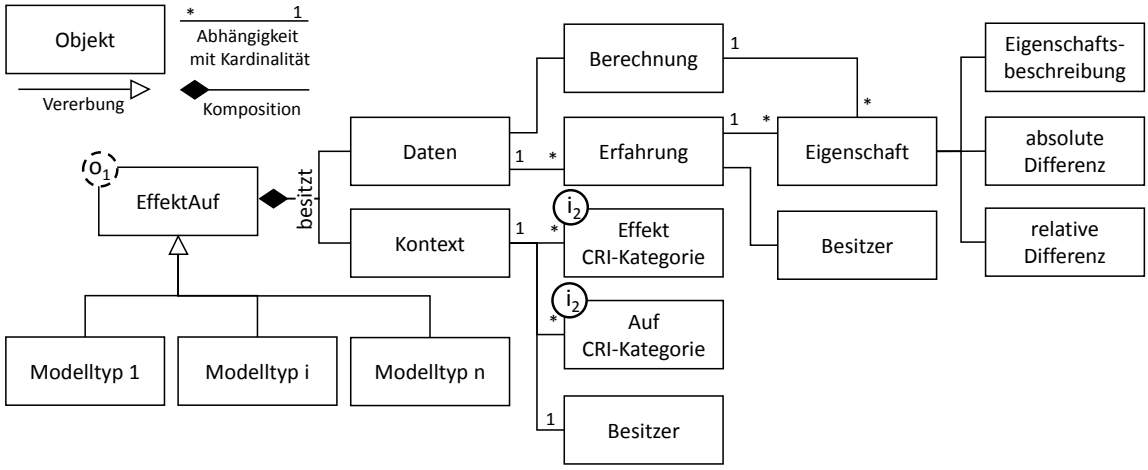


Abbildung 3.22: Datenstruktur des *EffektAuf*-Wertetypobjekts

Entsprechend ergibt sich die logische Datenstruktur des *EffektAuf*-Wertetypobjekts, wie in Abbildung 3.22 gezeigt. Es enthält als Daten ein Objekt des Typs *Berechnung*, in welches die gezeigten Berechnungen für beliebig viele Eigenschaften eingehen. Die Eigenschaften werden durch eine Beschreibung (beispielsweise eine Identifizierung) und ihre absolute und relative Eigenschaftsdifferenz charakterisiert. Zusätzlich können beliebig viele *Erfahrungen* aus dem Evolutionsprozess hinzukommen, die vergleichbare Berechnungen von anderen CPSen sind und somit anderen Besitzern haben. Der Effekt und Auf-Kontext kann entsprechend durch den Nutzer anhand der Kategorien des CRI-Modells beschrieben werden. Die Anwendung des Verhaltens auf die Modelltypen von CPPSen und CBCPSen beschreiben jeweils die folgenden beiden Abschnitte.

CPPS-Modelle:

CPPS-Modelle sind explizit hinsichtlich ihrer Analysierbarkeit bezüglich nichtfunktionaler Eigenschaften des unterliegenden Produktionsprozesses modelliert worden. Diese Eigenschaften werden operationalisiert, um sie direkt auf Verhalten des CPPSs zu beziehen. Dazu wurde eine Vielzahl von Eigenschaften und Metriken im Rahmen der Forschungstätigkeiten zu dieser Arbeit durch eine Literaturrecherche in einem Anforderungskatalog von operationalisierbaren Eigenschaften für Fertigungssysteme zusammengestellt (siehe [LFHL13]).[22] Dieser Katalog umfasst insbesondere Maße und Metriken zum Messen der *Performanz*, *Zuverlässigkeit* und *Flexibilität* der verwendeten CPPS-Modelle. Die Performanz bewertet die Leistung des CPPSs in Form von zeit- oder anzahlbezogenen Metriken. Beispiele sind hier der *Durchsatz*, der *Beleggrad* oder die *Effektivität*. Mit Zuverlässigkeit wird beschrieben wie verlässlich ein CPPS die ihm gestellten Aufgaben in einer definierten Zeitspanne durchführt. In Metriken kann dies beispielsweise durch das Messen der Fehler pro Zeitintervall beschrieben werden. Mit der Flexibilität wird angegeben, wie gut sich ein CPPS auf geänderte Bedingungen einstellen kann. Dies kann z.B. in Metriken bezüglich der Maschinen, aber auch des Transportsystems ausgedrückt werden.

Als Beispiel werden nachfolgend die Kennzahlen Produktionsrate und Routenflexibilität für Materialflussmodelle betrachtet.[23] Diese können nach Formel 3.31 für alle Werkstückausprägungen berechnet werden [SS90]. Zur Bestimmung sind als Messgröße die Anzahlen der Routen in einem CPS zu bestimmen [Lad18]. Die Produktionsrate ist nach Groover [Gro07] indirekt proportional zur Messgröße der mittleren Produktionsdauer eines Produktes und kann dementsprechend als gewichtetes Mittel für alle Werkstückausprägungen durch Formel 3.32 berechnet werden (vergleiche [Lad18]). Die Metrik für die Routenflexibilität gibt Formel 3.31 an (vergleiche [Lad18]).

$$f_{analyse}^{Routenflexibilität} = 1 - \frac{1}{\sum_{ausprägung \in m} n_{Routen}^{ausprägung}} \tag{3.31}$$

$$f_{analyse}^{Produktionsrate} = \frac{1}{\frac{\sum_{ausprägung \in m} t_{\mu}^{ausprägung} \cdot n_{Werkstücke}^{ausprägung}}{\sum_{ausprägung \in m} \cdot n_{Werkstücke}^{ausprägung}}} \tag{3.32}$$

Beide Metriken werden in Abhängigkeit von der Werkstoffausprägung berechnet und können auch für diese einzeln angegeben werden. Sowohl die Kennwerte der Routenanzahl $n_{Routen}^{ausprägung}$ als auch die mittlere Produktionsdauer $t_{\mu}^{ausprägung}$ werden anhand der Modellinstanz des Meta-Modells nach Verfahrensvorschrift 4 berechnet. Diese verwendet eine Tiefensuche, welche für alle Werkstoffausprägungen auf die verschiedenen Modellelemente des Meta-Modells für Materialflussmodelle angewendet wird.

Die Verfahrensvorschrift 4 bestimmt die Starttransitionen und Endstellen des Materialflusses und kennzeichnet alle Elemente als nicht besucht (weiß). Anschließend wird das iterative Grundprinzip nach Cormen et al. [CLRS13] auf die Besonderheiten der Modellelemente des Materialflussmodells angewendet. Die realisierte Tiefensuche baut von der Starttransition ausgehend einen Baum aus Transitionen und Stellen auf, welcher über die Assoziationen der *Verbindungen* (*StelleZuTransition* und *TransitionZuStelle*) aufgebaut wird. Die Verfahrensvorschrift geht iterativ bis zur letzten Endstelle des Materialflusses, da beim Erreichen dieser eine neue Route gefunden ist und die Kennwerte für diese Route berechnet werden können.

[22] Eine ausführliche Betrachtung aller Kennzahlen der Modelle gibt Ladiges [Lad18].
[23] Im Rahmen der Evaluation wird zusätzlich die Auslastung für Maschinenzustandsmodelle betrachtet.

Verfahrensvorschrift 4: Analyse zur Produktionsdauer und Anzahl von Routen von Materialflussmodellen

Eingabe: Materialflussmodell m mit *Element*-Elementen $e_1, \ldots, e_i, \ldots, e_n \in m$

Ausgabe: Mittlere Produktionsdauern t_μ^{metall}, $t_\mu^{plastik}$
Anzahl der Routen n_{Routen}^{metall}, $n_{Routen}^{plastik}$
Gesamtzahl $n_{Werkstücke}^{metall}$, $n_{Werkstücke}^{plastik}$

/* Suche für alle Werkstofftypen */

for *Werkstückausprägung* $ausprägung$ *in* m **do**

TIEFENSUCHE($ausprägung$, $n_{Werkstücke}^{ausprägung}$, $n_{Routen}^{ausprägung}$, $t_\mu^{ausprägung}$)

$t_\mu^{ausprägung} = \frac{t_\mu^{ausprägung}}{n_{Werstücke}^{ausprägung}}$

/* Tiefsuche für jeden möglichen Startpunkt */

function TIEFENSUCHE($ausprägung$, $n_{Werkstücke}^{ausprägung}$, $n_{Routen}^{ausprägung}$, $t_\mu^{ausprägung}$)

$farbe[\,]$, $ref[\,]$

$start[\,], ende[\,]$

/* Finde Start- und Endelemente über Zu/Von-Assoziationen */

for $e \in m$ **do**

if e **instanceof** $Transition$ **and** *hat keine* zu *Assoziation* **then**

$start[|start|] = e$

if e **instanceof** $Stelle$ **and** *hat keine* von *Assoziation* **then**

$end[|end|] = e$

$farbe[e] = weiss$

$ref[e] = null$

for $e \in start$ **do**

$ebene = 0$

BESUCHE(e, $ausprägung$)

Rückgabe $t_\mu^{plastik}$, $n_{Routen}^{plastik}$, t_μ^{metall}, n_{Routen}^{metall}

/* Besuche ein Element */

function BESUCHE(e, $ausprägung$)

$farbe[e] = grau$,$ebene^{++}$

/* Nachfolgendes Verbindung über Von/Zu-Assoziation */

if e **instanceof** *Transition* **then**

v = Von-Assoziation von Transition e

else

v = Zu-Assoziation von Stelle e

/* Besuche interativ Elemente, in die die Verbindung eingeht und ein Zeitverhalten für *ausprägung* besitzen */

for $e_{nachfolger}$ *hat eingehen-Assoziation zu* v **and** *Zeitverhalten für* typ **do**

$ref[e_{nachfolger}] = e$

BESUCHE($e_{nachfolger}$)

$farbe[e] = schwarz$,$ebene^{++}$

/* e ist Endstelle */

if $e \in ende[\,]$ **then**

$n_{Werkstucke}^{ausprägung} +=$ n des Zeitverhalten von e

$n_{Routen}^{ausprägung ++}$

/* Zurückverfolgen der Route */

for $i = ebene;\ i \geq 0;\ i^{--}$ **do**

$t_\mu^{Route} +=$ μ des Zeitverhalten von e

$t_\mu^{ausprägung} = t_\mu^{Route} \cdot n$ des Zeitverhalten von e

Die Route wird anschließend über eine *Array*-Variable zurückverfolgt und die Mittelwerte aufsummiert. Das Verfahren wird danach an der letzten Verzweigung fortgesetzt. Besuchte Transitionen und Stellen werden als grau und abgeschlossene als schwarz markiert. Das Verfahren berücksichtigt bei jeder Tiefensuche nur die Zeitverhalten einer Werkstoffausprägung. Entsprechend bedarf es der Annahme, dass das Materialflussnetz nach den Vorgaben des Generierungsverfahren korrekt ist. Dies beinhaltetet insbesondere, dass es für jede Ausprägung von Werkstücken eine Starttransition gibt und jede Endstelle mit einer Starttransition über einen vergleichbaren Pfad (siehe Abschnitt 3.5.2) verbunden ist. Weiterhin muss jede Transition oder Stelle auf einem solchen vergleichbaren Pfad liegen.

Mit den berechneten Kennwerten können Formel 3.31 und 3.32 folgend die Kennzahlen der Eigenschaften berechnet werden und die absolute und relative Differenz der Eigenschaften im Wertetypobjekt bestimmt werden.

CBCPS-Modelle:

Die gewählten CBCPS-Modelle von PCM sind insbesondere dafür konstruiert, eine Analyse auf Modellebene zu ermöglichen, um Performanzengpässe, Skalierungsprobleme oder die Wartbarkeit zu analysieren [BKR09]. Entsprechend können das Repositorymodell und Methodenverhaltensmodell zur Evaluation von verschiedenen nichtfunktionalen Eigenschaften während verschiedener Entwicklungsstadien genutzt werden [RBB+]. Dabei bieten die untersuchten CBCPS-Modelle Metriken für unterschiedliche Abstraktions- und Gruppenzugehörigkeiten (vergleiche dazu [Hof13]).

In der vorliegenden Arbeit werden insbesondere die Performanz nach der Definition von Rohr [Roh15] und die Wartbarkeit mit den Metriken des MOOSE (*Metric Suite for Object-Oriented Software Engineering*)-Katalogs von Chidamber und Kemerer [CK94] verwendet.[24]

Performanz umfasst alle Arten der Optimierung des Ressourcenverbrauchs. Dabei berücksichtigen die Methodenverhaltensmodelle insbesondere den Durchsatz, die Latenz und die Antwortzeit. Nachfolgend wird die Bestimmung der Antwortzeit vorgestellt. Diese ist definiert als die Dauer zwischen einer Anfrage und der entsprechenden Antwort [SW02]. Eine geringere Antwortzeit führt zu einer besseren Verfügbarkeit und ist ein guter Indikator für die Benutzerfreundlichkeit [HD00].

In dieser Arbeit wird die operative Antwortzeit von Dienstaufrufen betrachtet. Diese definiert sich als die Dauer zwischen Beginn und Ende einer Aufrufausführung - einschließlich der Zeit, die durch Folgeaufrufe innerhalb eines bearbeiteten Aufrufs stattfinden [Roh15]. Die in dieser Arbeit betrachtete Antwortzeit bewertet das CBCPS auf dem Abstraktionsniveau von Methoden und gehört zur Gruppe der Zeitverhaltensmetriken. Daraus ergibt sich die erste betrachtete Eigenschaft für CBCPSe indem Performanz über den Kehrwert der operativen Antwortzeit[25] des Methodenverhaltensmodells definiert wird:

$$f_{analyse}^{Performanz} = \frac{1}{t_{Antwortzeit}} \tag{3.33}$$

[24] Ein umfassender Metrikenkatalog wurde zusammen mit Paul Bischof im Rahmen seiner Bachelorarbeit [Bis16] zusammengestellt.

[25] gemessen in Schritten von 10 ms

Für die Wartbarkeit sind die Kohäsion und die Kopplung entscheidend [JBNK14]. Im Sinne der Arbeit ist Kohäsion dabei die interne Interaktion innerhalb eines Modells und die Kopplung die externen Aufrufe des Modells. Da die Beobachtung auf externem Verhalten basiert, wird nachfolgend die Kopplung betrachtet. Eine geringe Kopplung macht das betrachtete CBCPS dabei modularer und damit besser wiederverwendbar, sowie Änderungen des Softwarecodes weniger anfällig für Seiteneffekte [CK94]. Eine geeignete Metrik ist nach dem MOOSE-Katalog [CK94] die Kopplung zwischen Objekten (*coupling between objects*).

$$f_{analyse}^{Wartbarkeit} = \frac{1}{n_{Kopplungen}} \tag{3.34}$$

Diese Kopplungsmetriken können für Methodenverhaltensmodelle auf der Methodenebene angewendet werden. Eine *Kopplung* $m \rightleftharpoons \tilde{m}$ besteht in Methodenverhaltensmodellen genau dann, wenn das Modell der Methode m über ein *ExternerAufruf*-Element eine andere Methode $\tilde{m}$ aufruft oder durch diese aufgerufen wird. Daraus ergibt sich die zweite betrachtete Eigenschaft der Wartbarkeit. Diese wird in Formel 3.34 über den Kehrwert der Kopplung innerhalb des Methodenverhaltensmodells definiert.

Verfahrensvorschrift 5: Analyse von Methodenverhaltensmodell

Eingabe: Methodenverhaltensmodell m mit *Aktion*-Elementen $a_1, \ldots, a_i, \ldots, a_n \in m$
Ausgabe: Mittlere Antwortzeit t_μ, Menge der Kopplungen $kopplungen_{aufrufen}$

$ebene = 0$
BESTIMMEZEIT($m, ebene$)
Rückgabe t_μ, $kopplung_{aufrufen}$

```
/* Suche beginnend von StarayrtAktion                                    */
```

function BESTIMMEZEIT($m, ebene$)
- $a_{momentan}$ =StartAktion von m
- **while** *Nachfolger-Assoziation von* $a_{momentan}$ **do**
  - **if** $a_{momentan}$ **instanceof** *InterneAktion* **then**
    - t_μ += μ von $bedarf$ des BenötigterKontext von $a_{momentan}$

```
/* Iteration des Aufrufes                                                */
```

  - **else if** $a_{momentan}$ **instanceof** *ExternerAufruf* **then**

```
/* Neue Kopplung hinzufügen                                              */
```

    - $name$ = methodenname der aufrufen-Assoziation von $a_{momentan}$
    - **if** $kopplungen_{aufrufen}$ **not contains** $name$ **then**
      - füge $name$ in $kopplungen_{aufrufen}$ ein
    - m_{Aufruf} = Methodenverhaltensmodell zur aufrufen-Assoziation von $a_{momentan}$
    - $ebene$++
    - t_μ += BESTIMMEZEIT($m_{Aufruf}, ebene$)

```
/* Weiterbetrachtung der Verzweigungsverhalten                           */
```

  - **else if** $a_{momentan}$ **instanceof** *VerzweigungsAktion* **then**
    - p =Wahrscheinlichkeit von $a_{momentan}$
    - m_1 = Erste Transition-Assoziation von $a_{momentan}$
    - $t_\mu^{Iterativ1}$ = BESTIMMEZEIT($m_1, ebene$)
    - m_2 = Zweite Transition-Assoziation von $a_{momentan}$
    - $t_\mu^{Iterativ2}$ = BESTIMMEZEIT($m_2, ebene$)
    - $t_\mu = \frac{\mu^{Iterativ1} \cdot p + t_\mu^{Iterativ2} \cdot (1-p)}{2}$
    - **Exit** While-Schleife
  - **else if** $a_{momentan}$ **instanceof** *EndAktion* **then**
    - **Exit** While-Schleife

Die Verfahrensvorschrift 5 bestimmt die, für die Eigenschaften notwendigen, Kennwerte anhand einer Modellinstanz von Methodenverhaltensmodellen. Diese Kennwerte sind die Antwortzeit und die Menge von aufgerufenen Methoden. Dafür werden zunächst ausgehend von dem *StartAktion*-Element über die Vorgänger- und Nachfolgebeziehungen des Modells die Zeittupel der *BenötigterKontext*-Elemente aufsummiert. Wobei ein *ExternerAufruf*-Element iterativ aufgelöst werden kann indem rekursiv die Methode BESTIMMEZEIT(m) aufgerufen wird.[26] Methoden ohne ein *ExternerAufruf*-Element haben nur ein einziges *InterneAktion*-Element mit ihrem Zeitkontext. Wenn eine Verzweigung vorliegt, werden die beiden hinterlegten Verhaltensmodelle iterativ weiter betrachtet und die Ergebnisse entsprechend des Wahrscheinlichkeit-Attributes gewichtet. Dies folgt der Idee des PCM-Simulators der unterliegenden *RDSEFF*-Modelle. Das Endresultat ergibt die Antwortzeit t_μ, sowie den Zeitkontext der betrachteten Methode.

Als Annahme muss das Methodenverhaltensmodell nach den Vorgaben des Generierungsverfahrens korrekt sein. Das heißt, dass ein vergleichbarer Pfad aller Aktionen mit Vor- und Nachfolger, sowie nur eine *Startaktion*- und eine *Endaktion*-Element bzw. eine Verzweigung vorhanden sein muss (siehe Abschnitt 3.5.2).

Zusätzlich werden durch die Verfahrensvorschrift auf der untersten Aufrufhierarchie der betrachteten Ursprungsmethode alle Elemente des Typs *ExternerAufruf* in einer Menge gespeichert und als Ergebnis zurückgegeben. Diese Menge $kopplung_{aufrufen}$ enthält alle Kopplungen, welche das Modell der Methode m selbst vornimmt. Zusätzlich muss noch bestimmt werden, durch welche Methodenverhaltensmodelle aller weiteren Methoden m_i ein Aufruf von m vorgenommen wird. Dazu wird Verfahrensvorschrift 5 für jede Methode m_i durchgeführt. Dadurch ergibt sich die Kopplungen aller Aufrufe an das Modell der betrachteten Methode:

$$kopplung^{m}_{aufgerufen} = \{m_1, \cdots, m_i, \cdots, m_k\} \; für \; die \; gilt \; m \in kopplung^{m_i}_{aufrufen} \tag{3.35}$$

Daraus ergibt sich die endgültige Menge aller Kopplungen als Vereinigungsmenge und $n_{Kopplungen}$ als die gesuchte Kennzahl:

$$n_{Kopplungen} = |kopplung_{aufrufen} \cup kopplung_{aufgerufen}| \tag{3.36}$$

Mit den berechneten Kennzahlen können die absolute und relative Differenz der Eigenschaft im Wertetypobjekt gesetzt werden. Diese werden zusammen mit einem etwaigen expliziten Kontext als Ergebnis im Evolutionsschritt verwendet.

Zusammenfassung der Auswirkungsbestimmung

Zusammenfassend können durch die vorgestellten Verfahren Kennzahlen von Eigenschaften aus den Modellen extrahiert werden. Dabei stellt die automatische Analyse nach nichtfunktionalen Eigenschaften weitergehend einen passenden Ansatz dar, um Versionen und Varianten von koevolvierenden Modellen zu bewerten. Dafür werden definierte Kennzahlen über ihre absolute und relative Differenz in Beziehung gesetzt. Über diese Analyse kann die Auswirkungen einer Änderung mit Eigenschaften quantifiziert werden. Dadurch wird der Nutzer in die Lage versetzt, abzuschätzen welche Effekte durch die Anwendung eines Evolutionsschrittes in seinem CPS zu erwarten sind.

[26] Es wird angenommen das keine Zyklen von Aufrufen im CBCPS vorliegen.

3.5.5 Kategorisierung des Grunds über Abweichungen

Der unterliegende Grund der Abweichung ist automatisiert nur schwer abzuleiten, da er durch den *Black-Box*-Ansatz einer externen Beobachtung nicht direkt erfasst wird, sondern nur die Verhaltensänderung und deren Auswirkungen beobachtet werden können. Daher ist für den Grund der explizit gegebene Kontext bezüglich des CRI-Modells von größerer Bedeutung als beim Ergebnis oder der Reaktion. Wobei an dieser Stelle betont sei, dass eine Evolutionsunterstützung immer auch ohne eine explizite Benennung des Grunds erfolgen kann.

Dennoch gibt es in dieser Arbeit jeweils einen bestimmbaren Grund für einen Evolutionsschritt. Denn eine Evolution wird im Sinne der Arbeit immer durch ein nicht modellkonformes Verhalten ausgelöst. Diese Abweichung leitet, sofern sie als gewollt angenommen wird, eine Koevolution der Artefakte ein, welche jeweils in einem Evolutionsschritt mündet. Entsprechend ist im Sinne der Arbeit diese Abweichung der Grund für den Evolutionsschritt. Diese kann genutzt werden, um zu bewerten in welcher Situation sich das evolvierte CPS befand.

Abbildung 3.24 zeigt die für Gründe verwendete Datenstruktur. Diese enthält den Vergleich des beobachteten Ereignisses e des Ereignis-Streams und dem erwarteten Ereignis $\tilde{e}$ der Ereignisfolge des Modells. Dies geht konform mit der Definition einer Abweichung, welche nach Chandola [CBK09] ein beobachtbares Muster von Daten ist, das nicht konform zu dem erwarteten Verhalten ist. Dabei sind die möglichen Abweichungen von den unterliegenden Datenstrukturen abhängig. Dennoch lassen sich über Steams von Ereignissen allgemeine domänenunabhängige Abweichungen definieren. Für die Arbeit werden die allgemeinen Fehlerfilter für Streams nach Junker [Jun16] auf Klassen für Abweichungen von Ereigniskanälen der Modellartefakte übertragen:

- **Wegfallen:** Ein Ereignis in einem Streams tritt nicht mehr auf. Es wird statt eines erwarteten Ereignisses e, das leere Ereignis $\square$ im Stream durch das Modellartefakt beobachtet. Diese Klasse tritt auch oft bei Fehlern der Hard- oder Software des CPSs auf. Da die Abweichung allerdings als gewollt angenommen wurde, ist zumeist eine Komponente oder Element nicht mehr im CPS vorhanden bzw. die Abfolge bei der Nutzung dieser wurde verändert.
- **Einfügung:** Es wird ein zusätzliches Ereignis im Stream beobachtet. Durch das Modellartefakt wird das leere Ereignis $\square$ erwartet aber es tritt ein zu diesem Zeitpunkt nicht erwartetes Ereignis e auf.
- **Veränderung:** Ein erwartetes Ereignis tritt ein, besitzt aber eine dem Modell konträre Ausprägung. Das Modell erwartet somit Ereignis e, es tritt allerdings ein Ereignis $\tilde{e}$ auf, für das $e \neq \tilde{e}$ gilt. Beispielsweise unterschiedet sich der Typ- oder Ortskontext des Ereignisses signifikant.
- **Zeitaberration:** Ein durch das Modellartefakt erwartetes Ereignis e tritt auf, allerdings ist es bezüglich seines Zeitkontextes unterschiedlich. Dies kann ein zu frühes oder zu spätes Eintreffen sein. Bezüglich der Ausprägung besteht Gleichheit zwischen dem erwarteten und beobachteten Ereignis.
- **Übergreifend:** Zusätzlich können auch Abweichungen auftreten, die über ein einzelnes Ereignis des Streams hinausgehen. Dies sind insbesondere modellspezifische Abweichungen, wie die Veränderung von berechneten Parametern. Diesen ist gemeinsam, dass neben der bloßen Beobachtung des Streams zusätzliche Mechanismen zur Bestimmung von Abweichungen vorhanden sein müssen. Abweichungen dieser Klasse haben zumeist eine komponentenübergreifende Änderung als Grund.

Mit diesen allgemeinen Kategorien ergibt sich das Modellverhalten des Grundartefakts, wie in Abbildung 3.23 gezeigt. Dazu werden die Abweichungen aufgenommen und entsprechend kategorisiert. Diese werden mit dem etwaigen expliziten Kontext als *TriggerVon*-Wertetypobjekt transferiert.

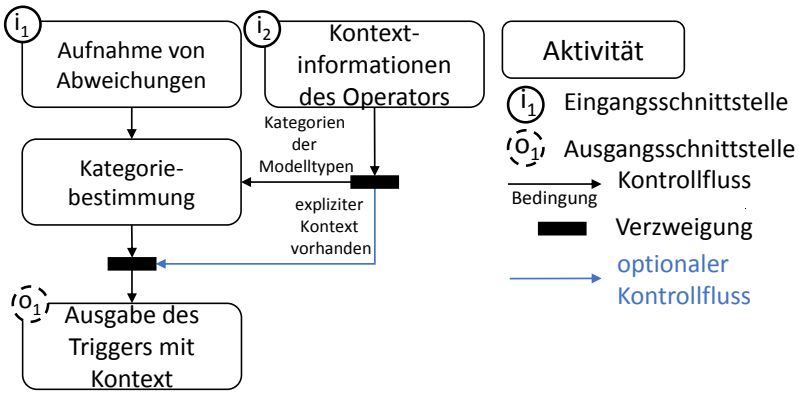


Abbildung 3.23: Artefaktverhalten zur Kategorisierung von Abweichungen

Das Wertetypobjekt besteht aus den erwarteten Ereignissen und den beobachteten Ereignissen der Ereigniskanäle sowie aus der Abweichungsklasse abw (siehe Abbildung 3.24). Neben dem Besitzer des Grunds kann der bereits angesprochene explizite Kontext in den Kategorien des CRI-Modells spezifiziert werden und dem Wertetypobjekt beigefügt werden. Nachfolgend werden die Klassen für die Abweichungen der einzelnen Modelltypen konkretisiert.

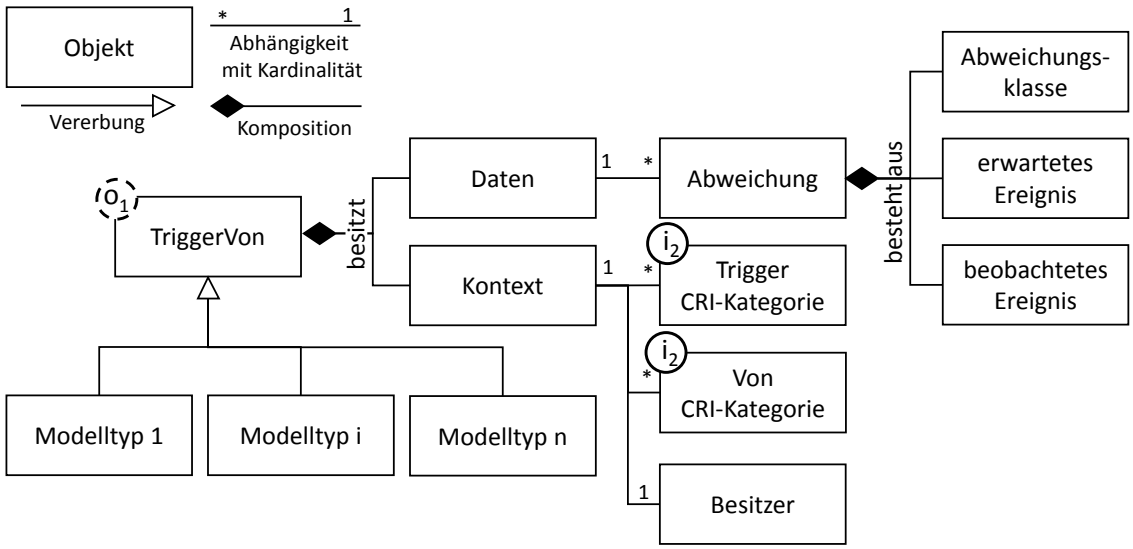


Abbildung 3.24: Datenstruktur des *TriggerVon*-Wertetypobjekts

CPPS-Modelle:

Bei CPPS-Modellen werden Abweichungen durch die Stimulation der Modelle nach der Semantik von Petri-Netzen erkannt. Zusätzlich dazu wird der Zeitkontext von Stellen und Transitionen bezüglich signifikanter Über- oder Unterschreitungen beobachtet. Nachfolgend werden die erfassbaren Abweichungen der Modelle, sowie ihre häufigsten dafür verantwortlichen Gründe angegeben:[27]

- **Wegfallen:** Unter diese Klasse fällt eine Abweichung, wenn ein oder mehrere kombinierte Signalereignisse nicht beobachtet werden, obwohl die zugehörige Transition feuerbereit ist. Dabei betrachten Materialflussmodelle nur materialflussrelevante, steigende Flanken und Maschinenzustandsmodelle beide Flanken in den Ereignissen. Dies lässt darauf schließen, dass ein Element des CPSs, wie ein Sensor oder Aktor, während der Evolution entfernt wurde oder nicht mehr Teil des abgebildeten Verhaltens ist.
- **Einfügung:** Diese Klasse von Abweichungen tritt auf, wenn ein, durch das Generierungsverfahren, als relevant eingestufte Ereignis auftritt, obwohl keine Transition feuerbereit ist. Während der Evolution wurde bei dieser Klasse oft ein CPPS-Element hinzugefügt, oder das CPPS anderweitig erweitert.
- **Veränderung:** In diese Klasse wird eine Abweichung eingeordnet, wenn ein, durch das Generierungsverfahren, als relevant eingestuftes Ereignis auftritt, dieses aber nur eine Unter- bzw. Übermenge einer feuerbereiten Transition ist. Zumeist wurde als Grund der Abweichung im abgebildeten CPPS ein Austausch, eine Erweiterung oder eine Reduktion vorgenommen.
- **Zeitaberration:** Ein spezifiziertes (kombiniertes) Ereignis tritt im Stream auf, besitzt allerdings eine signifikante Abweichung bezüglich seines Zeitverhaltens. Dies kann sowohl für Stellen als auch Transitionen auftreten. Der Grund der Evolution ist dabei oft eine (nichtfunktionale) Änderung an dem CPPS.
- **Übergreifend:** Als einzige übergreifende Funktionalität überwachen Materialflussmodelle die Produktionsverhältnisse an ihren Transitionverzweigungen. Diese geben Auskunft darüber ob beispielsweise der Produktionsmix des CPPSs verändert wurde.

CBCPS-Modelle:

Beim Vergleich von Nachrichtensequenzen des Informationsartefakts mit den generierten CBCPS-Modellen entstehen Abweichungen, wenn ein Nachrichtenpaar eines Aufrufs oder eine Teilsequenz dieser Paare in der Nachrichtensequenz existieren, die zu ihrer Konformität neue oder veränderte Entitäten, Assoziationen oder Attribute erfordern. Dazu werden die Nachrichten sequenziell überprüft indem für jede Nachricht der Aufruf im Methodenverhaltensmodell verfolgt wird sowie die Existenz der aufgerufenen Dienste und Komponenten im Repositorymodell mit den Kontextinformationen überprüft wird. Dabei können folgende Abweichungen entsprechend der allgemeinen Klassifizierung entstehen:

- **Wegfallen:** Es existieren im Aufrufpfad *ExternerAufruf*-Elemente die noch nicht in der Nachrichtensequenz und damit als Ereignis im CPS beobachtet wurden. Oder ein Dienst oder Komponente wird nicht mehr in den Kontextpaaren der Nachrichten aufgeführt. Dies deutet darauf hin, dass die Methode bzw. Komponente oder Dienst durch eine Reduzierung oder Umstrukturierung des CBCPSs entfernt wurde oder nicht mehr genutzt wird.

[27] Die Klassen entsprechen in Teilen den nach [Lad18, LFL16, LHFL15] erkennbaren Anomalien von Materialflussmodellen und Maschinenzustandsmodellen.

- **Einfügung:** In einem Aufrufpfad oder in den Kontextinformationen ist ein zusätzlicher Aufruf einer Methode bzw. eine zusätzliche Komponente oder Dienst enthalten. Das CPS wurde dann vermutlich in der Evolution um die entsprechende Funktionalität erweitert.
- **Veränderung:** Ein erwarteter Methodenaufruf hat veränderte Kontextinformationen oder andere Ausführungsinformationen. Diese Klasse von Abweichungen dient als Indikator dafür, dass eine entsprechende Evolution an vorhandenen CBCPS-Elementen vorgenommen wurde.
- **Zeitaberration:** Das Intervall zwischen Start- oder Endzeit von Aufrufen einer Methode ist verändert. Dies deutet darauf hin, dass der Quellcodes der Methode verändert wurde oder z.B. neue Methodenaufrufe hinzugefügt wurden.
- **Übergreifend:** In dieser Arbeit wird nur das Verhältnis von getätigten Aufrufen bei Verzweigungen innerhalb einer Methode betrachtet. Dies lässt auf ein durch die Evolution verändertes Verhalten schließen.

Zusammenfassung der Kategorisierung von Abweichungen

In diesem Abschnitt wurden verschiedene Kategorien für allgemeine Abweichungen definiert, die in den Ereignis-Streams auftreten können. Diese können in Abhängigkeit von der Ausprägung des CPSs spezifiziert werden. Durch die Abweichungen kann in Teilen auf den Grund einer Änderung geschlossen werden. Wenn keine Abweichung vor einer Änderung aufgetreten ist, ist diese entsprechend auch nicht zu klassifizieren. Eine Abweichung wird im Rahmen dieser Arbeit jedoch immer angenommen.

Die klassifizierte Abweichung versetzt einen Nutzer in die Lage, zu bewertet aus welchem Grund eine Evolution vorgenommen wurde und ob ein solcher Grund in ähnlichen Komponenten ebenso vorliegt bzw. vorliegen kann. Beim Grund ist dabei die Einführung eines expliziten Kontextes für den Evolutionsschritt durch den Nutzer besonders sinnvoll.

3.5.6 CPSe als Besitzer von Evolutionsschritten

Unter der avisierten Evolutionsunterstützung dieser Arbeit soll Wissen in verschiedenen Artefakten gespeichert, verarbeitet und ausgetauscht werden. Dadurch soll für CPSe, die solche Artefakte implementieren, ein objektives Bewusstsein entwickelt werden. Dieses objektive Bewusstsein wird in Kapitel 4 thematisiert und softwaretechnisch umgesetzt. Aus methodischer Sicht sind Daten, Informationen, Wissen als auch Erfahrung deshalb an das CPS gebunden. Um diesen Aspekt von Evolutionsschritten zu modellieren, wird neben dem Typ-, Ort- und Zeitkontext für alle Wertetypobjekte auch der Besitzer als Kontextobjekt modelliert.

Wie die Kategorien von CPSen zeigen (vergleiche Abschnitt 2.4.3), besteht in CPSen eine enge Interaktion mit dem Nutzer. Dieser installiert, verwaltet, steuert oder beobachtetet das CPS. Dennoch strebt diese Arbeit an, dass die Daten, Informationen und das Wissen an das CPS selbst gebunden werden, weshalb der Nutzer nur Teil des Kontextes des Besitzers ist.

Abbildung 3.25 zeigt die verschiedenen Ausprägungen des, in einem Artefakt explizit gemachten, Besitzers. Der Besitzer ist dabei ein Systemteil, welches das gesamte CPS sein kann, aber auch nur ein Teil dieses. CPSe bestehen dabei aus CPS-Komponenten, die weiterhin aus einer oder mehrerer hierarchischen Ebenen von Elementen zusammengesetzt sein können. Dies sind beispielsweise Methoden oder Sensoren und Aktoren des CPSs. Die Systemteile sind als Besitzer durch einen Identifikator und Namen gekennzeichnet.

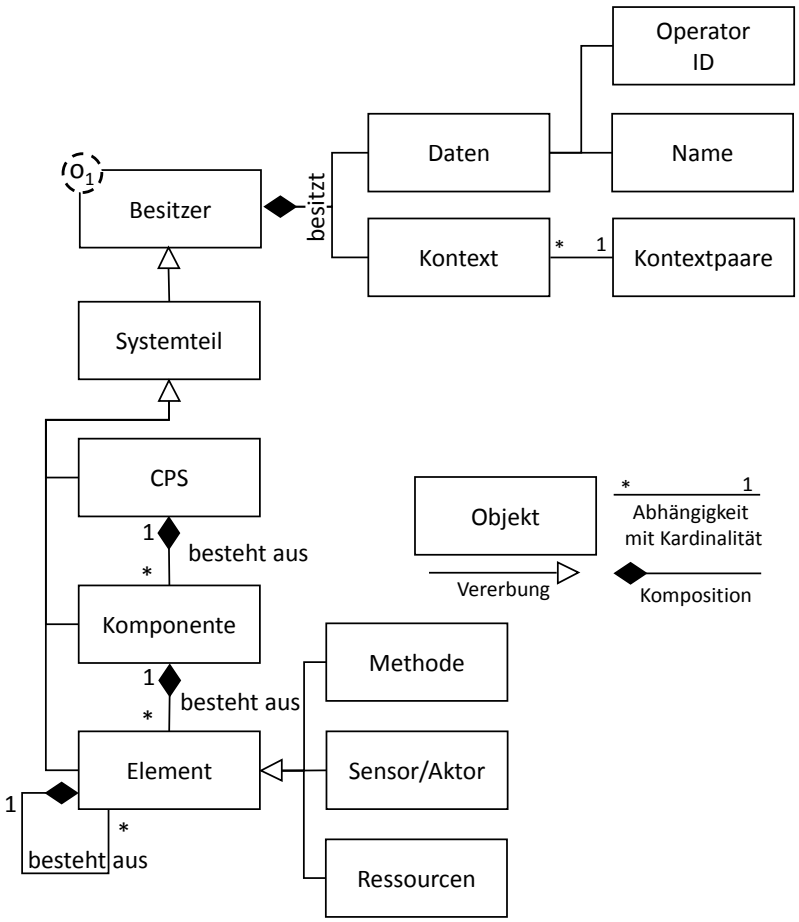


Abbildung 3.25: Datenstruktur des *Besitzer*-Wertetypobjekts

Zur Bestimmung des Besitzers ist kein besonderes Verfahren notwendig. Denn wenn ein Modell das CPS oder eine CPS-Komponente abbildet, ist diese(s) direkt Besitzer der daraus abgeleiteten Evolutionsschritte. Je nach dem betrachteten Modell können somit Systemteile der niedrigeren oder höheren Systemebenen als Besitzer von Evolutionsschritten auftreten.

Der Kontext wird halbautomatisiert vom Nutzer durch den Kanal $c^{O}_{MetaInfo}$ definiert. Dieser umfasst neben dem Nutzer des CPSs auch die Systemteilhierarchie, an der der Besitzer im CPS verortet werden kann. Der Kontext wird nur bei der Explizitmachung im Besitzerartefakt gesetzt, nicht jedoch bei den vorherigen Wertetypobjekten (*Ereignis*, *Information*, *KontextInfo*, *Modell*, *TriggerVon*, *WirkungAn* und *EffektAuf*). Eine Unterscheidung in CPPS und CBCPS ist bezüglich des Besitzers dabei nicht notwendig.

3.5.7 Zusammenführung in Evolutionsschritten

Mit Grund, Reaktion, Ergebnis und Besitzer wird die konzeptionelle Beschreibung eines Evolutionsschrittes vorgenommen. Der Evolutionsschritt beschreibt damit in unterschiedlichen Aspekten wie ein Modell m in ein folgendes Modell $\tilde{m}$ evolviert ist. Das Evolutionsschrittartefakt wird nachfolgend als $\Delta(m, \tilde{m})$ notiert.[28]

[28] Im Rahmen der Evaluation wird zum besseren Verständnis als Index des Modells die Konfiguration der Fallstudie notiert.

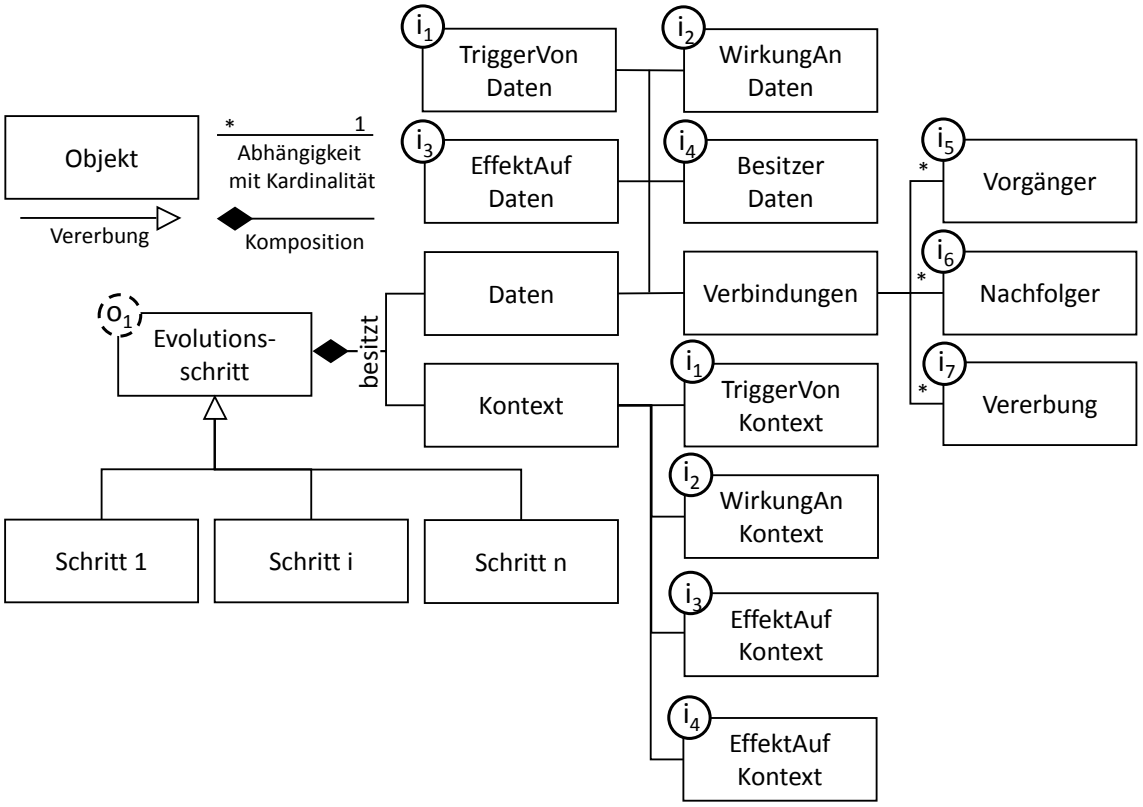


Abbildung 3.26: Datenstruktur des *Evolutionsschritt*-Wertetypobjekts

Das durch das Evolutionsschrittartefakt transferierte Wertetypobjekt besteht aus den Daten und Kontexten der einzelnen beteiligten Artefakte. Abbildung 3.26 zeigt, dass die Daten und Kontexte der durch die Eingangsschnittstellen i_1, i_2, i_3, i_4 erhaltenen Wertetypobjekte *TriggerVon*, *WirkungAn*, *EffektAuf* und *Besitzer* im Evolutionsschritt enthalten sind. Dies gilt auch für die entsprechenden Kontexte. Mit diesen Elementen wird ein grundlegender Evolutionsschritt beschrieben. Dieser wird immer dann gebildet, wenn eine neue Version eines Modells vorliegt.

Zusammenfassung der Erstellung von Evolutionsschritten

Zusammenfassend kann durch die Artefakte des Grunds, der Reaktion, des Ergebnisses und des Besitzers die wesentlichen konzeptionellen Artefakte eines Evolutionsschrittes aus den Versionen von Modellen abgeleitet werden. Diese beschreiben jeweils unterschiedliche konzeptionelle Aspekte des Evolutionsschrittes. Aus jedem dieser Aspekte kann der Nutzer Schlussfolgerungen über die unterliegende Änderung ableiten. Dafür stehen ihm u.a. eine modellbasierte Beschreibung der Verhaltensänderung (Reaktion), die erwarteten Eigenschaftsveränderungen durch diese Änderung (Ergebnis) und eine Klassifizierung des als nicht konform bewerteten Verhaltens vor der Änderung (Grund) zur Verfügung.

Es bleibt festzustellen, dass die Evolutionsschrittartefakte aus koevolvierenden Modellartefakten für diese Arbeit als zentrale Beschreibungsform im Sinne der ersten und zweiten Hypothese dienen.

3.6 Evolutionsunterstützung durch Prozessabbildung und Vorschläge

Die dritte Hypothese des methodischen Kapitels besagt, dass Evolutionsschritte auch komponentenübergreifend in einem CPS eingesetzt werden können, um den Nutzer bezüglich der Dokumentation und Realisierung von Evolution zu unterstützen. Dies wird nachfolgend für Evolutionsschritte gezeigt. Dazu wird zum einen die Abbildung eines Evolutionsprozesses untersucht und zum anderen die Ableitung von Vorschlägen.

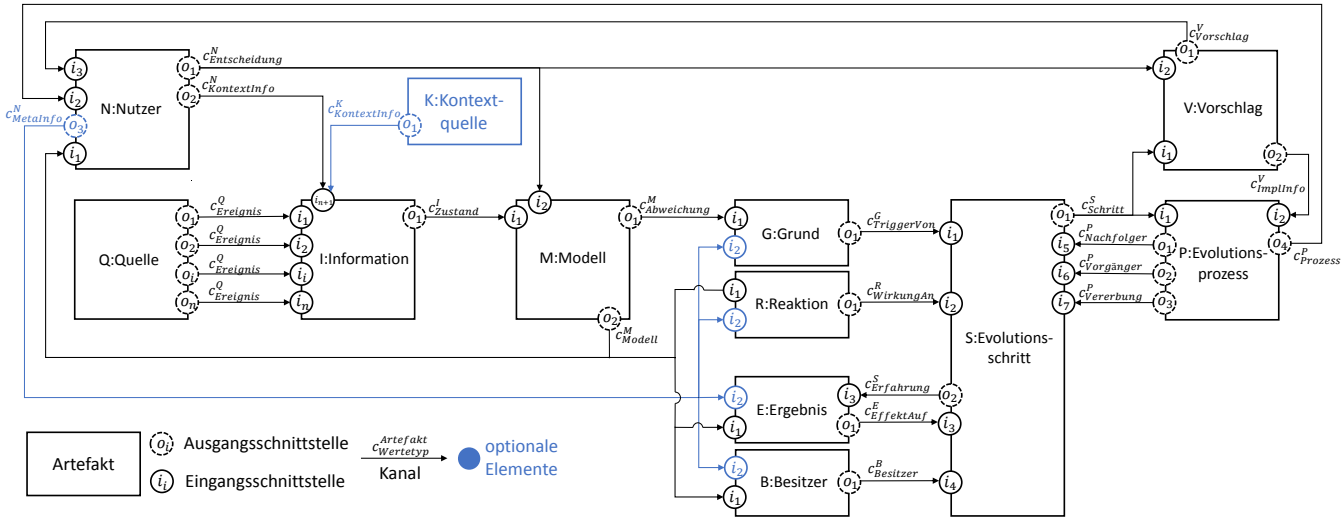


Abbildung 3.27: Artefaktstruktur für die Dokumentation eines Evolutionsprozesses und Erstellung von Vorschlägen

Die Vorgänger und Nachfolger des Evolutionsprozesses werden, wie in Abbildung 3.27 zu sehen, durch eine Relation im *Evolutionsprozessartefakt* gebildet. Dazu werden Evolutionsschritte über Relationen geordnet (Abschnitt 3.6.2). Diese Vorgänger und Nachfolger werden über die entsprechenden Kanäle $c^P_{Nachfolger}$ und $c^P_{Vorgänger}$ an das *Evolutionsschrittartefakt* transferiert.

Basierend auf den geordneten Evolutionsschritten und dem aktuellen Modell des Nutzers wird ein Vorschlag zur Fortführung von Evolution abgeleitet, der in einem Vorschlagsartefakt realisiert wird (Abschnitt 3.6.1). Der Vorschlag stellt eine Handlungsalternative bereit. Diese prognostiziert für den Nutzer, wie sich das konkrete aktuelle Verhalten des CPSs mit der Anwendung des Evolutionsschrittes verändert. Dies umfasst insbesondere eine modellbasierte Vorhersage des Verhaltens sowie eine Abschätzung der dadurch veränderten Eigenschaften des CPSs.

Dieser kann den Vorschlag anschließend Implementieren, was nicht Teil der Betrachtung in dieser Arbeit ist. Die Implementierung des Vorschlags durch den Nutzer mündet allerdings in einer erneuten Evolution, deren Evolutionsschritt im Evolutionsprozess als Vererbungsrelation zum Evolutionsschritt des Vorschlags abgebildet wird (Abschnitt 3.6.2). Wenn ein Evolutionsschritt vererbt wird, wird dieser dem Evolutionsprozess und anschließend dem ursprünglichen Evolutionsschrittartefakt über Kanal $c^P_{Vererbung}$ mitgeteilt.

Der Nutzer agiert bei dieser Evolutionsunterstützung hauptsächlich als Entscheidungsträger über Vorschläge sowie ihre Implementierung. Zur Unterstützung erhält er dabei neben den Modellen zur Dokumentation des Systemverhaltens (c^M_{Modell}), eine Übersicht der Evolutionsschritte im Evolutionsprozess ($c^P_{Prozess}$) und Vorschläge zu dessen Fortführung ($c^V_{Vorschlag}$).

3.6.1 Unterstützung der Evolution durch Vorschläge

Als zentrales Artefakt der Fortführung der Evolution dient dabei das *Vorschlagsartefakt*. Für dessen Artefaktverhalten sind die drei grundlegenden Aspekte, d.h. der jeweilige Grund, die Reaktion und das Ergebnis zu betrachten. Dabei muss die **Anwendbarkeit** des Evolutionsschrittes anhand seiner unterschiedlichen Artefakte bewertet werden. Dazu wird der Trigger des Grunds bewertet, die Wirkung auf das vorhandene aktuelle Modell angewendet sowie der entsprechende Effekt abgeschätzt. Mit diesen Indikatoren für eine erfolgreiche Anwendung wird eine Bewertung des Evolutionsschrittes berechnet und bewertet, ob der Evolutionsschritt dem Nutzer als ein möglicher Evolutionsschritt vorgeschlagen wird.

Abbildung 3.28 zeigt das daraus resultierende Wertetypobjekt des Vorschlagsartefakts. Dieses besitzt zwei wesentliche Datenobjekte: erstens den ursprünglichen Evolutionsschritt und zweitens das durch die Anwendung der Wirkung der Reaktion erzeugte Modellinstanz, die als **Handlungsalternative** bezeichnet wird. Zusätzlich enthält der Vorschlag die berechnete Bewertung, die als Indikatoren für die Anwendbarkeit der Handlungsalternative dienen.

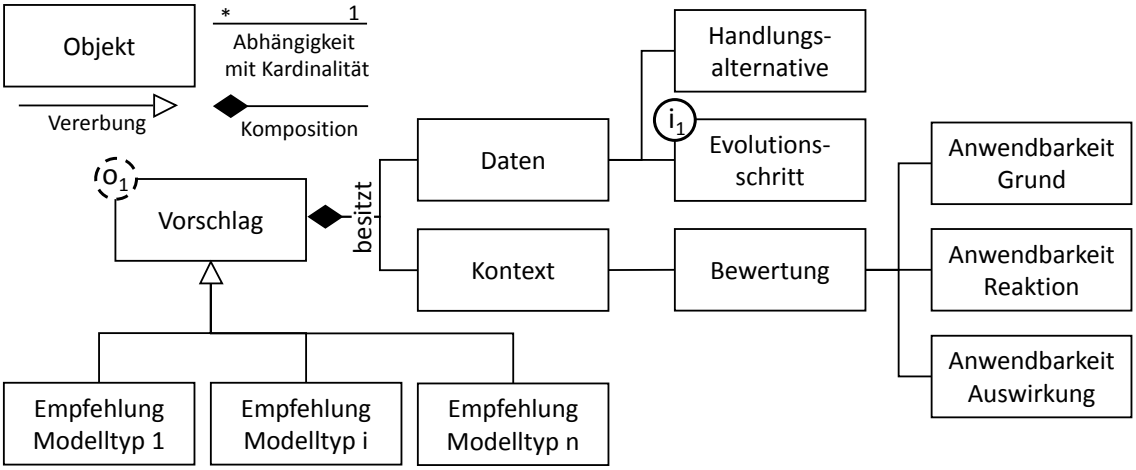


Abbildung 3.28: Datenstruktur des *Vorschlag*-Wertetypobjekts

Wenn die Handlungsalternative durch den Nutzer in einer Implementierung umgesetzt wird und somit der Vorschlag angenommen wird, wird dieses über den Entscheidungskanal $c^N_{Entscheidung}$ mitgeteilt. Die dadurch entstehende Vererbung wird genauso wie die zeitlichen und räumlichen Relationen des Evolutionsschrittes im Evolutionsprozess abgebildet und in Abschnitt 3.6.2 näher erläutert.

Ableitung eines Vorschlags

Für die dritte Hypothese einer vollständige Evolutionsunterstützung müssen Evolutionsschritte auch auf vergleichbare CPSe und somit auf Versionen und Varianten von Modellen angewendet werden, sofern deren Kontext vergleichbar ist. Editierskripts, die zusammen mit dem Modellkontext die Wirkung der Reaktion bilden, können für diese Modelltransformation verwendet werden [Kel10, Keh15]. Dazu liegen die Operationen der Wirkung in einer Halbordnung vor, die nach der Parameterauflösung auf ein Modell angewendet werden kann. Wie bereits bei der Erzeugung der Wirkung beschrieben, müssen dafür alle Eingangsparameter, die nicht durch eine in der Halbordnung vorhergehende Operation erzeugt wurden, an Modellelemente des neuen Modells gebunden werden.

Diese für die resultierende Handlungsalternative wesentliche Anpassung der Modelltransformation wird durch Verfahrensvorschrift 3 der Zuordnung geleistet. Denn statt einer Zuordnung zwischen zwei Modellen, muss hierbei eine Zuordnung eines Modells $\hat{m}$ und den nicht belegten Eingangsparametern des Modellkontextes $\delta(m,\tilde{m})_{kontext}$ vom ursprünglichen Modell m stattfinden. Abhängig davon, ob im Modellkontext das gesamte Modell m oder nur die Kontexte der Modellelemente der Eingangsparameter vorliegen, kann dies durch die in Abschnitt 3.5.2 definierten Ähnlichkeiten bestimmt werden: Wenn das gesamte Modell im Modellkontext vorliegt, können die Folgen der Eingangsparameter e aus dem Editierskript $\delta(m,\tilde{m})$ verwendet werden. Mit diesen Folgen, die durch den Modellkontext $\delta(m,\tilde{m})_{kontext}$ gegeben sind, kann dann eine Zuordnung $z(\delta(m,\tilde{m}),\hat{m})$ mit dem neuen Modell $\hat{m}$ bestimmt werden. Diejenigen Modellelemente $\hat{e} \in \hat{m}$, für die $e \rightarrow \hat{e}$ in dieser Zuordnung enthalten ist, werden in der Anwendung des Editierskripts dann an die jeweiligen Eingangsparameter gebunden. Wenn nur die Kontexte von e und keine seiner Folgen vorliegen, wird für jedes Modellelement $\hat{e}$ aus $\hat{m}$ der Substitutionswert aus Formel 3.13 verwendet und dabei der jeweils höchste Wert als Zuordnung verwendet. Bei identischen Werten zwischen mehreren Zuordnungen wird ein Gleichungssystem analog zum linearen Gleichungssystem in Formel 3.19 gelöst.

Das Resultat der Modelltransformation bei Anwendung des Editierskripts unter Verwendung des Modellkontextes ist dann die Handlungsalternative des Vorschlags $\hat{m}^{\Delta(m,\tilde{m})}$. Der Vorgang kann wie folgt definiert werden:

$$\hat{m} \rightarrow^{\delta(m,\tilde{m}),z(\delta(m,\tilde{m}),\hat{m})} \hat{m} \circ m\delta\tilde{m} = \hat{m}^{\Delta(m,\tilde{m})} \tag{3.37}$$

Bewertung von Vorschlägen

Um Vorschlägen zu bewerten, wird eine **Bewertungsfunktion** des Nutzers verwendet. Dabei werden an dieser Stelle einzig die Datenteile des Evolutionsschrittes verwendet, damit eine automatisierte Bewertung möglich bleibt. Die Kontextinformationen, welche manuell durch den Nutzer gegeben werden können, sind im Ansatz der Arbeit optional, weshalb sie nicht verwendet werden sollen. Dennoch können über die Kontexte beliebige Erweiterungen für die Bewertung, beispielsweise durch eigene Klassifizierungen, vorgenommen werden.

Die beispielhafte Bewertung in dieser Arbeit basiert auf einer beliebig gewichteten Kombination einer Abschätzung von mehreren Anwendungsbewertungen. Diese werden isoliert für den Grund, die Reaktion und das Ergebnis bestimmt. Nachfolgend wird methodisch gezeigt, wie eine beispielhafte Bewertungsfunktion für alle drei betrachteten Aspekte eines Evolutionsschrittes definiert werden kann:[29] diskutiert.

- Das wesentliche Kriterium des Grunds sind die Abweichungen, die bei der parallelen Beobachtung von Modell und CPS aufgetreten sind. Das Kriterium hat dabei dem Datenmodell des *TriggerVon*-Wertetypobjekts (siehe Abbildung 3.24) folgend drei wesentliche Faktoren: das beobachtete Ereignis e und das erwartete Ereignis $\tilde{e}$, sowie die Klasse der Abweichung abw nach der in Abschnitt 3.5.5 gegebenen Klassifizierung. Die Präferenzen bei der Bewertung sind durch den Nutzer zu geben, der damit bestimmte Arten von Abweichungen präferieren kann. Während die Bewertung der Klassen direkt spezifiziert werden, wie beispielsweise $bewertung^{abw}_{Zeitaberration} = 0,5$, wird die Bewertung von Ereignisausprägungen über Regeln auf den Kontext definiert. Dabei wird für jede Ereignisausprägungen die Bewertung 0 angenommen, sofern diese durch keine Regel anders spezifiziert ist. Eine im Rahmen von CPPSen verwendete Regel wäre beispielsweise

[29] Die gebotene Unterstützungsleistung wird anhand von unterschiedlichen Szenarien der Evolution im Rahmen der operativen Betrachtung in Kapitel 4

$bewertung^{e}_{(typ,\ Aktor-Werkstück-Halten)} = 1$. Diese Regel sagt aus, dass Abweichungen, in denen das erwartete Ereignis den Typkontext *Aktor-Werkstück-Halten* besitzt, einen erhöhten Anwendungsbewertungen haben.

Die im Rahmen dieser Arbeit verwendete Metrik für die Anwendungsbewertung des Grunds nimmt eine, durch einen Faktor k_{abw} gewichtete, Verstärkung der Klasse einer Abweichung vor, da diese im Rahmen der Evaluation als am bedeutendsten identifiziert wurde. Folgende Metrik wird für den Grund verwendet:

$$f_{Grund} = \frac{k_{abw} \cdot bewertung^{abw} + bewertung^{e} + bewertung^{\tilde{e}}}{k_{abw} + 2 \; mit \; k_{abw} \in \mathbb{Q}_{+}} \tag{3.38}$$

- Für die Reaktion gilt, dass eine Wirkung genau dann gut anwendbar ist, wenn die Ähnlichkeit des Modellkontextes und die des vorliegenden Modells für alle Parameter des Editierskripts der Wirkung hoch ist. Diese lässt sich aus der Zuordnung bestimmen, wie sie im vorherigen Abschnitt erläutert wurden. Die Anwendungsbewertung der Reaktion ist für die Arbeit definiert als normierte Summe der Ähnlichkeiten aller Zuordnungen:

$$f_{Reaktion} = \frac{\sum_{f_{Ähnlichkeit}(e_i, \tilde{e}_{\tilde{i}})}}{|z(\delta(m, \tilde{m})_{kontext}, \tilde{m})|} \; \forall e_i \leftrightarrow \tilde{e}_{\tilde{i}} \in z(kontext_m, \tilde{m}) \tag{3.39}$$

- Um die Anwendungsbewertungen des Ergebnisses zu bestimmen, werden die Eigenschaftsveränderungen des Evolutionsschrittes bezüglich der relativen Differenz betrachtet. Wie in der Datenstruktur des *EffektAuf*-Wertetypobjekts gezeigt (siehe Abbildung 3.22), besteht diese Eigenschaftsdifferenz aus einer Berechnung für den aktuellen Schritt und Erfahrungen für Vererbungen des Schrittes. Für die vorliegende Arbeit ist diese Berechnung durch Verfahrensvorschrift 6 realisiert.

  Diese Verfahrensvorschrift benötigt als Eingabe die Eigenschaftsmengen der Berechnung und der Erfahrungen als Array von n Eigenschaften. Es wurde für die Arbeit Gewichtungen der Eigenschaften ($\omega_{es_1}, \cdots, \omega_{es_i}, \cdots, \omega_{es_n}$) verwendet, wodurch die, durch eine obere Grenze max beschränkten, Eigenschaften priorisiert werden können. Unter den Gewichten wird die Höhe der Bewertung des Schrittes für jede Eigenschaftsmenge als gewichtetes, normiertes Mittel berechnet. Dabei werden positive und negative Eigenschaftsdifferenzen miteinander aufgewogen.[30] An dieser Stelle kann, je nach Eigenschaften, eine eigene, beliebige Funktion zur Berechnung verwendet werden. Um über Berechnung und Erfahrungen einen festen, normierten Wert $f_{Ergebnis} \in [0, 1]$ für die Anwendungsbewertung zu bestimmen, werden die Bewertungen aller Eigenschaftsmengen aufsummiert und normiert. Hierbei wird ein reduziertes Gewicht für Erfahrungen verwendet ($k_{Erfahrung}$), da diese nicht direkt berechnet wurden, sondern von einer anderen Komponente stammen. Das Resultat ist die Bewertung des Schrittes bezüglich des Ergebnisses.

[30] Ein solches Vorgehen nimmt an, dass Eigenschaften für die Bewertung austauschbar sind und setzt voraus das die Bewertung der Eigenschaften durch eine hohe positive, relative Differenz auch einen positiven Nutzen für den Nutzer haben. Dies ist in dieser Arbeit durch Formel 3.30 gegeben, da die betrachteten Eigenschaften positiv sind. Alternative Vorgehen sind ebenso möglich.

Verfahrensvorschrift 6: Berechnung der Anwendungsbewertungen

Eingabe: Berechnung des Schrittes $bArray[\,] = \{relDiff_{es_1}, \cdots, relDiff_{es_i}, \cdots, relDiff_{es_i}\}$
Erfahrungen des Schrittes
$erfahrung = \{eArray_{e_1}[\,], \cdots, eArray_{e_i}[\,], \cdots, eArray_{e_n}[\,]\}$
mit $eArray_e[\,] = \{relDiff_{es_1}, \cdots, relDiff_{es_i}, \cdots, relDiff_{es_n}\}$

Ausgabe: Anwendungsbewertungen des Ergebnisses $bewertung_{Ergebnis}$

$k_{erfahrung} = 0,5$

/* Bewertung der Berechnung */

$bewertung_{Berechnung} +=$ BEWERTEN($bArray[\,]$)

/* Bewertung der Erfahrung */

$bewertung_{Erfahrung} = 0$

for $eArray_e[\,] \in erfarung$ **do**

  $bewertung_{Erfahrung} +$ BEWERTEN($eArray_e[\,]$)

$$bewertung_{Ergebnis} = \frac{bewertung_{Berechnung} + k_{erfahrung} \cdot bewertung_{Erfahrung}}{1 + |erfahrung| \cdot k_{erfahrung}}$$

Rückgabe max$(bewertung_{Ergebnis}, 0)$

/* Bewertung einer Eigenschaftsmenge */

funtion BEWERTEN($esArray[\,]$)

  max = 1 /* Maximum der Differenz */

  $esSumme$ = 0 /* Summe der Bewertungen */

  for $relDiff_{es} \in esArray[\,]$ **do**

    if $relDiff_{es} \geq 0$ **then**

$$esSumme = esSumme + \frac{\mathbf{min}(relDiff^{es_i}_{\tilde{m}-m}, max)}{|esArray|}$$

    else

$$esSumme = esSumme + \frac{\mathbf{max}(relDiff^{es_i}_{\tilde{m}-m}, -max)}{|esArray|}$$

  Rückgabe $esSumme$

Die Bewertungsfunktion $f_{bewertung}$ wird als gewichtetes Mittel der Anwendungsbewertungen von Grund, Reaktion und Ergebnis des Evolutionsschrittes $\Delta(m, \tilde{m})$ definiert. Für die Bestimmung muss dabei sowohl der Evolutionsschritt $\Delta(m, \tilde{m})$ als auch das Modell $\hat{m}$ für die Bewertungsfunktion verfügbar sein.

Mit der Bewertung eines Evolutionsschrittes wird entschieden, ob der Evolutionsschritt dem Nutzer vorgeschlagen wird. Wenn die Bewertung über einem Schwellwert $\Theta_{Vorschlag}$ liegt, wird der entsprechende Evolutionsschritt vorgeschlagen. Wenn $\Theta_{Vorschlag} < f_{bewertung}$ gilt, wird diese Handlungsoption aufgrund eines zu kleinen erwarteten Nutzens für den Nutzer verworfen und nicht vorgeschlagen.

Die exemplarisch gezeigte Berechnung der Bewertung eines Evolutionsschrittes ist methodisch nur eine von mehreren möglichen Definition, welche für diese Arbeit versucht, eine möglichst allgemeingültige Bewertung für Vorschläge zu definieren. Denn alle so berechneten Bewertungen sind jeweils als fall- und modellspezifisch zu betrachten und sollten deshalb manuell durch den Nutzer genauer spezifiziert werden, um eine aussagekräftige konkrete Bewertung zu erhalten. Dies wird als Alternative im Rahmen eines Ähnlichkeitsmaßes für Varianten in der Evaluation (Abschnitt 6) gezeigt.

Zusammenfassung der Bewertung

Durch die Ableitung und Bewertung von Evolutionsschritten können Handlungsalternativen abgeleitet werden. Diese enthalten die Abweichung, die zur Änderung geführt hat, eine

vollständigen modellbasierte Verhaltensbeschreibung der vorgenommenen Änderung, sowie deren vermutete Auswirkungen auf nichtfunktionale Eigenschaften. Die damit gewonnenen Handlungsalternativen werden nutzerspezifisch bewertet und ausgewählt, damit der Nutzer eine fundierte Entscheidung zur Fortführung der Evolution treffen kann. Dies entspricht der in der dritten Hypothese formulierten Evolutionsunterstützung.

3.6.2 Evolutionsprozess als Artefakt

Dem artefaktbasierten Evolutionsprozess dieser Arbeit folgend gibt es zwei unterschiedliche Verbindungen zwischen Evolutionsschritten: Zum einen ist dies der Zusammenhang von Versionen und Varianten, wobei Versionen die zeitliche Verknüpfung und Varianten die räumliche Verknüpfung abbilden. Die andere Verbindung beschreibt die Wiederverwendung von Evolutionsschritten. Diese Verbindung zwischen zwei Evolutionsschritten entsteht, wenn basierend auf einem abgeleiteten Vorschlag, eine neue Version implementiert wird. Der aus dieser neuen Version abgeleitete Evolutionsschritt steht zu dem ursprünglichen Evolutionsschritt, der für den Vorschlag genutzt wurde, in Relation.

Die wesentlichen Daten der Wertetypobjekte sind dabei eine Relation der Vererbung und eine Relation von Vor- und Nachfolgern. Diese werden durch Paare abgebildet, welche jeweils zwei Evolutionsschritte miteinander verknüpfen. Wenn eine neue Verknüpfung stattfindet, werden im Evolutionsprozessartefakt die Relationen aktualisiert und die entsprechenden Evolutionsschritte über die Ausgangsschnittstellen o_1, o_2 oder o_3 informiert (vergleiche Artefaktverhalten in Abbildung 3.27).

Evolutionsprozessrelation des Raums und der Zeit

Die Erzeugung dieser Relation basiert auf den Modellbeschreibungen der Evolutionsschritte, durch welche gleiche Modellinstanzen identifiziert werden können. Dabei gilt, dass eine Vorgängerrelation zwischen zwei Evolutionsschritten besteht, wenn das Ursprungsmodell dem Folgemodell entspricht. Für die Nachfolgerelation gilt dies entsprechend umgekehrt. Für die Relation zweier Evolutionsschritt $\Delta_1(m_1, \tilde{m}_1)$ und $\Delta_2(m_2, \tilde{m}_2)$ gilt somit:

$$\begin{aligned} &\textbf{if } m_1 = \tilde{m}_2 \textbf{ then } \Delta_1(m_1, \tilde{m}_1) \ \succ_{ZR} \Delta_2(m_2, \tilde{m}_2) \\ &\textbf{else if } \tilde{m}_1 = m_2 \textbf{ then } \Delta_2(m_2, \tilde{m}_2) \ \prec_{ZR} \Delta_1(m_1, \tilde{m}_1) \end{aligned} \tag{3.40}$$

Die zeitlich-räumliche Verknüpfung wird durch die definierte Nachfolgerelation $\succ_{ZR}$ definiert. Diese Nachfolgerelation bildet eine partielle Ordnung, indem der gesamte Evolutionsprozess des CPSs, von dem jeder Evolutionsschritt ein Teil ist, eine *passierte-vorher*-Relation (*happend-before*; vergleiche [CDK05]) wie bei Kausalordnung von Ereignissen in asynchronen verteilten Systemen aufspannt.

Diese injektive Nachfolgerelation ist auf die Menge aller Evolutionsschritte S_Δ definiert. Wobei ϕ die Wurzel des durch die Relation aufgespannten Baumes ist, der alle initialen Entwicklungen des CPSs, also Evolutionsschritte ohne definierte Vorgänger, nachfolgen.

$$\succ_{ZR}: S_\Delta \cup \phi \rightarrow S_\Delta \; mit \; S_\Delta = \{\Delta_1, \cdots, \Delta_i, \cdots, \Delta_n\} \tag{3.41}$$

Aus Formel 3.41 folgt, dass es sich bei der durch die Relation erzeugen Ordnung um eine strikte partielle Ordnung handelt, welche irreflektiv und antisymmetrisch ist. Diese Ordnung hat dabei das minimale Element ϕ, da per Definition gilt:

$$\forall \Delta \in S_\Delta : \Delta \succ_{ZR} \phi \tag{3.42}$$

Die funktionale Vorgängerrelation $\prec_{ZR}$ ist die inverse Relation zur Nachfolgerelation $\succ_{ZR}$ und in Formel 3.43 definiert:

$$\prec_{ZR}: S_\Delta \rightarrow S_\Delta \cup \phi \; mit \; S_\Delta = \{\Delta_1, \cdots, \Delta_i, \cdots, \Delta_n\} \tag{3.43}$$

Es gilt somit die Äquivalenz $\succ_{ZR}^{\smile} \equiv \prec_{ZR}$, wobei $\prec_{ZR}$ genauso irreflektiv und antisymmetrisch ist und das maximale Element ϕ besitzt.

Durch diese Relationen werden Vorgänger und Nachfolger von Evolutionsschritten bestimmt. Wenn mehr als ein Nachfolger existiert, ein Evolutionsschritt in der Relation $\succ_{ZR}$ also mehrere Partner besitzt, dann wurde eine Variante erzeugt. Alle Nachfolger bzw. alle Vorgänger von Evolutionsschritten können über die transitive Hülle der Relationen $\succ_{ZR}$ und $\prec_{ZR}$ bestimmt werden. Diese transitive Hülle stellt den ersten Teil des Evolutionsprozesses dar.

Evolutionsprozessrelation der Vererbung

In der Relation der Vererbung kann jeder beliebige Evolutionsschritt vererbt werden, allerdings jeder Evolutionsschritt nur durch einen Evolutionsschritt erben. Dies hat den Grund, dass in der verwendeten Ableitung der Vorschläge keine Kombinationen von Evolutionsschritten betrachtet werden. Eine Vererbung liegt genau dann und nur dann vor, wenn der Vorschlag durch den Nutzer implementiert wurde und entsprechend das Evolutionsprozessartefakt ein Werteobjekt über die Eingangsschnittstelle i_2 erhalten hat. Dies impliziert, dass durch die Implementierung des Evolutionsschrittes das Modell ein verändertes Verhalten aufweist, welches durch Abweichungen des Modells erkannt wird. Die Vererbung ist entsprechend als eine injektive Relation $\succ_{VV}$ definiert:

$$\succ_{VV}: S_\Delta \rightarrow S_\Delta mit \; S_\Delta = \{\Delta_1, \cdots, \Delta_i, \cdots, \Delta_n\} \tag{3.44}$$

Die Relation besitzt kein minimales Element. Allerdings gibt es für alle nichtleeren Bäume ein kleinstes Element, das den ursprünglichen Evolutionsschritt, von dem geerbt wurde, darstellt. Für jedes kleinstes Element Δ_{min} gilt, dass keine Vererbung existiert:

$$\nexists \Delta \in S_\Delta : \Delta \succ_{VV} \Delta_{min} \tag{3.45}$$

Durch die Relation werden somit nichtzusammenhängende Vererbungsbäume mit dem ursprünglichen Evolutionsschritt als Wurzel aufgebaut. Die transitive Hülle der Relationen $\succ_{VV}$ stellt den zweiten Teil des Evolutionsprozesses dar.

Zusammenfassung der Prozessabbildung und Bewertung

Mit der räumlich-zeitlichen Verknüpfung wird über fest definierte Relationen ein Evolutionsprozess erzeugt, der von einem einzigen Element ausgeht und Versionen und Varianten eines CPSs abbilden kann. Über eine zusätzliche Vererbungsrelation kann die Verbreitung von Wissen im CPS festgehalten werden. Durch diese Vererbung wird eine zusammenhängende Menge von Evolutionsschritten erzeugt, die Änderungen beschreiben, die auf eine ähnliche Art und Weise durchgeführt wurden. Durch die unterschiedlichen Modellkontexte dieser Evolutionsschritte kann ein Nutzer somit den für ihn passendsten Evolutionsschritt aus dieser Menge auswählen.

Durch die systematische Bewertung und Verbreitung von Evolutionswissen wird die Prozessdokumentation aus der dritten Hypothese bestätigt. Denn durch die Relationen zwischen den Evolutionsschritten wird ein vollständiger, systemübergreifender Evolutionsprozess dokumentiert.

3.7 Zusammenfassung

Im methodischen Kapitel[31] wurden drei Hypothesen formuliert, um zu untersuchen, wie Evolution durch Artefakte systematisch abgebildet, koevolviert und fortgeführt werden kann.

Zur Untersuchung der ersten Hypothese einer allgemeinen Beschreibungsform wurde in Abschnitt 3.2 zunächst das Konzept eines schrittbasierten Evolutionsprozesses vorgestellt, der Änderungen in sequenziellen und parallelen Evolutionsschritten abbildet. Mit der Beschreibung von Evolutionsschritten hinsichtlich von sechs aus der Literatur entnommenen Aspekten, können der Grund, die Reaktion und das Ergebnis von Änderungen systematisch erfasst und dokumentiert werden.

Um der zweiten Hypothese einer Koevolution von Artefakten zu genügen, wurden zunächst in Abschnitt 3.3 unterschiedliche Ereignisquellen, wie die Beobachtung von Bussystemen und Dienstaufrufen, sowie externen Spezifikationen, wie der Quellcode und Testmodelle, für die Evolution methodisch integriert. Darauf aufbauend wurden in Abschnitt 3.4 Modellartefakte vorgestellt, die es erlauben, sowohl die zur Evolutionsunterstützung neu entwickelten Modelltypen, als auch bereits existierende Modelltypen durch Verfahren der Generierung, der Beobachtung und der Analyse in die Evolutionsunterstützung zu integrieren. Dies erlaubt die avisierte Koevolution von aktuellem Evolutionswissen für unterschiedliche Ausprägungen von CPSen, wie CPPSen und CBCPSen.

Die dritte Hypothese bezieht sich auf die Dokumentation und Fortführung der Evolution über Evolutionsschritte. Mit der methodischen Entwicklung eines Evolutionsschrittartefakts wurde in Abschnitt 3.5 gezeigt, wie die unterschiedlichen Aspekte von Evolutionsschritten gebildet werden können. Dazu wurden Abweichungen für Gründe klassifiziert, das System als Besitzer spezifiziert und die Ergebnisse der Evolution durch operationalisierte Eigenschaften messbar gemacht. Die Verhaltensänderung wird als modellbasierte Reaktion durch geliftete Modelldifferenzen abgebildet, wodurch Änderungen für den Nutzer verständlicher werden. Die so operationalisierten Aspekte werden in einem übergreifenden Konzept von Evolutionsschritten zusammengefügt und darin semi-formal definiert.

Über die Anwendung der Modelldifferenzen konnten weiterhin Handlungsalternativen für eine Fortführung der Evolution abgeleitet werden (Abschnitt 3.6). Dies ermöglicht dem Nutzer Änderungen besser zu bewerten. Dafür erhält er neben einer Dokumentation des sich stetig verändernden Verhaltens, der Eigenschaftensausprägungen sowie der beobachteten Abweichungen seines CPSs, auch eine auf sein CPS zugeschnittene, modellbasierte Empfehlung mit daraus resultierenden Eigenschaftsveränderungen. Diese werden aus vergangenen Evolutionsschritten von ähnlichen CPSen abgeleitet. Durch die Definition von Relationen auf Evolutionsschritten kann der dadurch entstehende Evolutionsprozess zusätzlich in seiner zeitlichen-räumlichen Verknüpfung, als auch bei der Weitergabe von Erfahrungen dokumentiert werden.

[31] Zusätzliche konkrete Beispiele der Anwendung dieser Methodik werden im Rahmen der Evaluation (Kapitel 6) anhand der Fallstudien dieser Arbeit gezeigt.

4 Wissenstragende Komponenten für cyber-physische Systeme

Im folgenden Kapitel wird die operationale Fragestellung dieser Arbeit näher betrachtet. Dazu werden zunächst Hypothesen für die drei Verknüpfungsdimensionen aufgestellt. Diesen folgend wird in Abschnitt 4.2 die allgemeine Verknüpfungsdimension für spezifische CPSe erläutert. Die folgenden drei Abschnitte stellen dann eine Systemarchitektur für eine wissenstragende Komponente vor, die ein objektives Bewusstsein über die Evolution eines CPSs erzeugt. Im abschließenden Abschnitt 4.6 wird eine räumliche Verknüpfung unter der Sichtweise von Evolutionsschritten vorgenommen. Dazu wird gezeigt, wie wissenstragende Komponenten miteinander verbunden werden können, um das Informationsnetzwerk von CPSen für die Evolutionsunterstützung zu nutzen.

4.1 Hypothesen der operationalen Untersuchung

Dem methodischen Lösungsansatz eines artefaktbasierten Evolutionsprozesses inhärent ist die Annahme, dass dafür notwendige Evolutionsartefakte durch das CPS selbst verwaltet werden. Diese softwaretechnisch herzustellende ist Gegenstand des operativen Teils der Arbeit. Für diesen werden nachfolgend drei Hypothesen aufgestellt.

Der artefaktbasierte Evolutionsprozess betrachtet Evolutionsartefakte als zentrales Element, um Evolution zu unterstützen. Die Arbeit geht dabei davon aus, dass dafür eine Erweiterung auf mehreren funktionalen Ebenen eines CPSs notwendig ist. Dies umfasst die bedingte Beobachtung der technischen Verbindungen, die daraus gewonnene Wahrnehmung der Evolution und einen automatisierten Vergleich in einem Cyber-Netzwerk. Die dafür entwickelte Erweiterung eines CPSs wird nachfolgend als **wissenstragende Komponente** bezeichnet.

Durch die unterschiedlichen Ausprägungen von CPSen ist eine wissenstragende Komponente maßgeblich durch die Domäne, in der das CPS operiert, geprägt (vergleiche [WW18]). Der artefaktbasierte Evolutionsprozess abstrahiert durch die Verwendung von Modellen von dieser Domäne. Allerdings ist diese Abstraktion bei der Erweiterung eines konkreten CPSs nicht mehr gegeben. Denn um komplexe Anwendungssituationen zu erfassen, müssen CPSe auf operativer Ebene ein gewisses Maß an Wissen über das spezifische System besitzen [GB12]. Deshalb wird in dieser Arbeit untersucht, wie Domainmodelle dieses Wissen für Evolutionsschritte verwendbar machen können. Daraus ergibt sich die erste Hypothese der operativen Untersuchung:

- Die artefaktbasierte Sichtweise ermöglicht eine domänenspezifische Erweiterung des CPSs durch eine wissenstragende Komponente auf mehreren funktionellen Ebenen eines cyber-physischen Systems.

Wie in der Problemstellung identifiziert, haben CPSe in der Regel kein Bewusstsein über sich selbst (*self-awareness*). Deshalb muss ein solches Bewusstsein für die Evolution durch die zusätzliche wissenstragende Komponente hergestellt werden. Dadurch wird das CPS in die Lage versetzt, neue und veränderte Bedingungen, Fehler oder Probleme von sich aus zu erkennen. Dies geht einher mit der identifizierten Problemstellung einer Maximierung der Produktivität des Nutzers durch die Reduktion seiner Aufgaben bei gleichbleibender

Qualität (siehe [MKCC03]). Für eine Realisierung einer solchen Evolutionsunterstützung sind smarte Architekturen und Modelle notwendig [ZT16]. Die daraus resultierende Hypothese einer nichtinvasiven Systemarchitektur wird für die Arbeit wie folgt formuliert:

- Eine wissenstragende Komponente für CPSe kann unter einer geeigneten Systemarchitektur ein objektives Bewusstsein von Evolution nichtinvasiv realisieren.

Die Bedeutung und Vernetzung von CPSen nimmt steig zu [GB12]. Dabei werden CPSe zu unternehmenskritischen Systemen, die innerhalb des CPSs, aber auch mit dem Nutzer und anderen CPSen aktiv Informationen austauschen und untereinander kooperieren müssen [TBC+14]. Dies erfordert von CPSen die Fähigkeit, sich nicht nur autonom zu beobachten, kontrollieren und dokumentieren, sondern diese Informationen auch geeignet weiterzuverbreiten [TBC+14]. Lee et al. [LKY14] sehen dadurch einen Trend in zukünftigen industriellen CPSen, dass Informationen in Flotten gleicher CPSe stetig propagiert werden. Dadurch können sie ein Bewusstsein über problematisches Verhalten entwickeln [LKY14]. Entsprechend sollte das Bewusstsein über Evolution auch durch Informationen ähnlicher CPSe aufgebaut werden. Dies erlaubt es, Artefakte der Evolution zu analysieren und intelligenten Evolutionsentscheidungen zu treffen. Eine solche Intelligenz soll durch verknüpfte Evolutionsschritte realisiert werden und wird in der dritten Hypothese operativ untersucht:

- Durch Netzwerktechnologien können wissenstragende Komponente dahingehend verknüpft werden, dass sie durch einen Austausch von Evolutionsschritten den Nutzer bei Evolutionsentscheidungen unterstützen können.

4.2 Operative Evolutionsunterstützung in einer wissenstragenden Komponente

Im folgenden Abschnitt wird eine operative Evolutionsunterstützung für cyber-physikalische Systeme zunächst allgemein betrachtet. Dazu werden in Abschnitt 4.2.1 allgemeine funktionale Ebenen einer Systemarchitektur für CPSe erläutert. Diese beschreiben, wie ein objektives Bewusstsein für Evolution allgemein zu erreichen ist. Im zweiten Abschnitt 4.2.2 wird gezeigt, wie die im methodischen Teil als zentrales Artefakt entworfenen Evolutionsschritte auf spezifische CPSe zugeschnitten werden.

4.2.1 Wissenstragende Funktionalität

Als zentraler Aspekt betrachten CPSe explizit nicht nur Software, sondern auch die physischen Bedingungen des Systems. Durch die damit verbundene starke Interaktion von Hardware und Software ist es schwer ein gesamtheitliches Bewusstsein des CPSs über seine Evolution zu schaffen, da Wissen immer über die verschiedenen Disziplinen hinweg betrachtet werden muss [AAH+11, HCL+17].

Wie im Grundlagenkapitel dieser Arbeit erläutert, sind dafür mit dem selbstreflektierten und objektiven Bewusstsein zwei mögliche Ansätze denkbar. Insbesondere für bereits laufende CPSe ist ein objektives Bewusstsein besser geeignet, da er nicht in das System eingreift (vergleiche [RSLR15, LHFL14a, HLF18]). Dies legt auch die identifizierten Rahmenbedingungen eines CPSs nahe. Denn in der komplexen Dynamik von Software und Hardware in CPSen ist in der Praxis für Funktionalitäten, die nicht die Kernbereiche des CPSs betreffen, kein direkter Eingriff in den Code gewünscht (vergleiche [HLLF14b]). Zusätzlich hat ein objektives Bewusstsein den Vorteil, keinen disruptiven Paradigmenwechsel zu benötigen. Denn ein solcher könnte konträr zu bestehenden Systemen, Technologien, Arbeitsabläufen oder Best Practices

sein. Dadurch könnte die Akzeptanz einer Evolutionsunterstützung schwinden obwohl die Vorteile eines Evolutionsbewusstseins durch das CPS selbst im Allgemeinen erwünscht sind.

Trotzdem könnte auch im Rahmen dieser Arbeit ein selbstreflektiertes Bewusstsein eingesetzt werden. Dies gilt insbesondere dann, wenn unterliegende Paradigmen, wie beispielsweise in einigen agentenbasierten Architekturen, eine Selbstwahrnehmung von sich aus bereitstellen (vergleiche [KB15, SH16, SRB17]). In solchen Ansätzen wird ein Bewusstsein durch das kontrollierende CPS selbst aufgebaut indem diese verfügbaren Meta-Informationen inhärent und reflektiert zur Verfügung stellen. Dies schließt allerdings bereits genutzte CPSe, die dem angestrebten Paradigmenwechsel nicht folgen können, aus.

Um ein Bewusstsein für Evolution aus Sicht einer Systemarchitektur für CPSe einzuordnen, zeigt Abbildung 4.1 auf der linken Seite fünf funktionale Ebenen eines CPSs. Diese Ebenen entsprechen einer weit verbreitenden Definition nach Lee et al. [LBK14] und werden folgend auf den Themenkomplex der Evolution angewendet:

1. Auf der niedrigsten Ebene erlaubt es die Verbindung von technischen Elementen in einem Prozess (*connection*) genaue und zuverlässige Daten des **technischen Prozesses** zu erfassen.
2. Auf der Ebene der **cyber-physischen Verbindung** (*conversion*) werden relevante Informationen aus den Daten extrahiert und an cyber-physische Komponenten übergeben, um diesen ein Bewusstsein zu ermöglichen.
3. Auf der Ebene des **Cyber-Netzwerks** (*cyber-level*) liegen Informationen vor, die zwischen verschiedenen Komponenten ausgetauscht werden können. Dadurch können sich diese Komponenten miteinander vergleichen.
4. Auf Ebene der Kognition (*cognition*) werden hauptsächlich Cyber-Elemente verwendet, die gesammelte Informationen präsentieren, um eine Entscheidungsunterstützung zu ermöglichen.
5. Die Konfigurationsebene (*configuration*) agiert als Kontrollsystem, das die übergeordnete Steuerung übernimmt. Dadurch wird Selbstadaption oder -konfiguration in einem CPS ermöglicht.

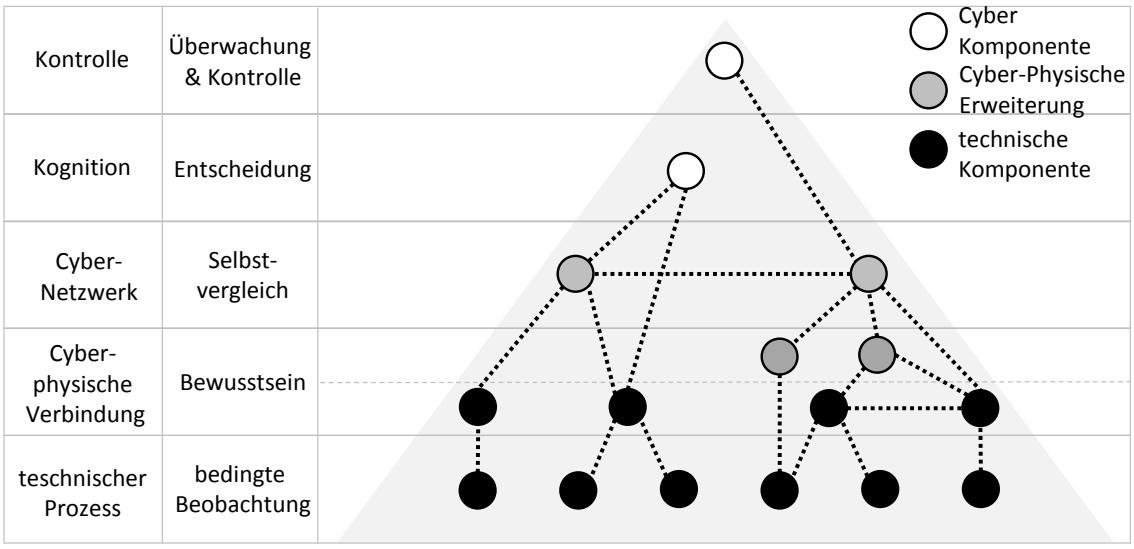


Abbildung 4.1: Ebenen und Fähigkeiten der Evolution in cyber-physischen Systemen

Die Ebenen von CPSen sind mit industriellen Systemen vergleichbar, die entlang der Automatisierungspyramide beschrieben werden [Mon14]. Dabei brechen CPSe jedoch mit deren Systemarchitektur. Denn CPSe sind, insbesondere auf den oberen funktionalen Ebenen, in verteilten und dezentralen Einzelkomponenten, wie Cloud-Diensten, unterteilt und werden aus diesen komponiert [Mon14]. Diese Komposition ist in Abbildung 4.1 auf der rechten Seite aus als Sichtweise dieser Arbeit dargestellt. Schwarze Kreise beschreiben Komponenten, die den technischen Prozess realisieren. Dieser ist in vielen industriellen Systemen ein realzeitlicher Prozess. In der cyber-physischen Verbindung wird dieser Prozess durch Sensoren, Aktoren und Controller abgebildet und gesteuert. Der technische Prozess ist dabei von sich aus schon interdisziplinär, was ihn schwerer als reine Softwaresysteme erfassbar macht [AAH+11, HLF18]. Für eine Evolutionsunterstützung muss der Zustand auf dieser Ebene kontinuierlich beobachtet und abgebildet werden [LBK14].

Zusätzliche cyber-physische Komponente (graue Kreise) können außerhalb des (realzeitlichen) Prozesses liegen. Diese dienen dazu, Funktionalitäten zu erfüllen, welche über die reinen technischen Anforderungen des CPSs hinaus gehen. Lee et al. [LBK14] bezeichnen die Funktionalität als Bildung eines Bewusstseins. In der vorliegenden Arbeit ist dies das Evolutionsbewusstsein. Dieses kann somit als Teil des CPSs entwickelt werden oder das CPS nachträglich erweitern, um beispielsweise Legacy-Systeme in CPSe zu integrieren.

Selbstwahrnehmende Softwareansätze, wie sie auf dieser funktionalen Ebene genutzt werden, basieren generell auf Verhaltensmodellen [KHV06]. Deshalb müssen die Modelle der Evolutionsmethodiken in dieser Ebene softwaretechnisch vollständig integriert werden. Aus Evolutionssicht abstrahiert ein solches Vorgehen von nicht abgebildeten Details der unterliegenden Ebenen des CPSs, da das CPS auf sein in den Modellen beobachtetes Verhalten reduziert wird. Allerdings werden dadurch die Informationen automatisiert erfassbar und liegen dem CPS inhärent und direkt vor.

Verhaltensmodelle, die wie in dieser Arbeit durch Beobachtung eines Systems generiert werden, sind auf ein A-posteriori-Verhalten beschränkt. Auskünfte über das sich fortentwickelnde CPS geben sie nicht, da eine Vorhersage durch die vorliegenden Informationen limitiert ist. Denn diese sind auf historische Informationen der bereits eingetretenen Situationen des betrachteten CPSs beschränkt, weshalb die Fortführung von Evolution auf das Fachwissen des Nutzers oder externe Kommunikationskanäle angewiesen ist [BS09]. Zusätzliche Informationen entstehen erst bei der Einbeziehung des Cyber-Netzwerks als Verteilungsebene des CPSs (siehe Abbildung 4.1). Dann wird es möglich, diese mit Informationen zu vergleichen, die nicht in direkter historischer Beziehung des CPSs stehen. Dazu sind die Informationen im Kontext der Ähnlichkeit zu vergleichbaren Informationen des Cyber-Netzwerks in Beziehung zu setzen. Dies wird von Lee et al. [LBK14] als Funktionalität des Selbstvergleichs bezeichnet. Im Rahmen dieser Arbeit sind dies Evolutionsschritte, welche auch die Vergleichbarkeit der Information umfassen. Entsprechend ist für diese Arbeit die Ähnlichkeit zweier CPSe durch die Ähnlichkeit ihrer Evolutionsschritte definiert.

Abbildung 4.1 zeigt weiterhin noch reine Cyber-Komponenten (weiße Kreise), die keine direkte Verbindung zu physischen Komponenten besitzen. Diese können weitere kognitive und konfigurierende Funktionalitäten bereitstellen. Wobei die auf dieser Ebene fokussierten Funktionalitäten in vielen CPSen und Ansätzen noch durch den menschlichen Nutzer abgedeckt sind. Auch in dieser Arbeit werden diese nicht vollständig automatisiert. Denn eine Entscheidung oder Selbstkonfiguration wird nicht angestrebt, sondern diese bleibt dem menschlichen Nutzer vorbehalten. Dennoch enthält die wissenstragende Komponente Methodiken, die Teilfunktionalitäten dieser Ebenen abdecken oder unterstützen.

Die wissenstragenden Komponenten agiert über diese funktionalen Ebenen eines CPSs hinweg. Dadurch entstehen drei wesentliche Informationsflüsse zwischen den Ebenen einer entsprechenden Architektur:

- Der erste Informationsfluss beschreibt die Bildung von Wissen im CPS. Dabei werden durch Beobachtung Daten und Informationen des technischen Prozesses auf Modellebenen der cyber-physische Verbindung und des Cyber-Netzwerks gehoben, die dann für die Evolutionsunterstützung softwaretechnisch vorgehalten werden.
- Der zweite Informationsfluss entsteht durch das Feedback zwischen dem Cyber-Netzwerk und den beiden obersten Ebenen des CPSs. Dabei ist dieser Informationsfluss bidirektional. Denn er umfasst zum einen das Feedback des Nutzers in Form von Entscheidungen und einer Konfiguration. Und zum anderen werden Evolutionsartefakte der mittleren funktionalen Ebenen an den Nutzer zurückgegeben, wie beispielsweise generierte Modelle.
- Der dritte Informationsfluss entsteht durch ein intelligentes Informationsmanagement innerhalb der cyber-physischen Verbindung und des Cyber-Netzwerks. Darin kommunizieren Komponenten untereinander und tauschen Informationen aus. Unter den identifizierten Hypothesen muss dieser Informationsfluss sowohl zeitlich als auch räumlich etabliert werden.

Die drei Informationsflüsse bestätigen den funktionalen Teil der ersten Hypothese, dass eine Erweiterung über die funktionalen Ebenen hinweg vorgenommen werden muss. Gemäß dieser Sichtweise wird eine wissenstragende Komponente für die Evolutionsunterstützung in CPSen damit wie folgt definiert:

> *Eine wissenstragende Komponente ist eine dem CPS zugehörige, softwaretechnische Einheit, die Informationen aus dem technischen Prozess extrahiert und diese in der Ebene der cyber-physischen Verbindung kombiniert und bewahrt sowie im Cyber-Netzwerk austauscht und vergleicht, um kognitive und konfigurierende Entscheidungen des Nutzer zu unterstützen.*

4.2.2 Evolutionsschritte für spezifische CPS

Nach den funktionalen Ebenen wird nun der domänenspezifische Teil der ersten Hypothese untersucht. Dazu sind Evolutionsschritte bezüglich ihrer Ausprägung für spezifische CPSe zu charakterisieren. Dies wird anhand der Anwendungsgebiete von CPPSen und CBCPSen gezeigt. Dazu wird zunächst für beide Domänen ein generisches Domänenmodell vorgestellt und es werden anschließend aus diesem Domänenmodell die Kategorien der Dimensionen (Grund, Von, Reaktion, An, Ergebnis, Auf) des CRI-Modells abgeleitet.[1]

Das CRI-Modell und damit die Beschreibung von Evolutionsschritten hängt vom Ziel der Evolution ab. Dieses kann nach dem Evolutionsverständnis der Arbeit (siehe Abschnitt 3.2.1) zum einen eine *verändernde Evolution* oder *erhaltende Evolution* sein. Eine verändernde Evolution will aktive Veränderungen des Systems bewirken, wie z.B. den Funktionsumfang erweitern. Eine erhaltende Evolution nimmt Veränderungen vor, die dem Zweck der Funktionserhaltung dienen. Dabei entsteht auch eine Schnittmenge, die Evolution beschreibt welche aktive Veränderungen anstrebt aber auch Aktivitäten umfasst, um unter diesen Änderungen den Funktionsumfang zu erhalten.

[1] Die Domänenmodelle basieren auf einer Literaturrecherche, den Fallstudien der Evaluation und verfügbaren Reports von Änderungen und Anomalien großer Softwareunternehmen.

Die unterschiedlichen Arten der Evolution werden nachfolgend berücksichtigt, um unter der ersten Hypothese zu untersuchen, inwieweit Evolutionsschritte auf spezifische CPSe zugeschnitten werden können. Dazu wird für CPPSe eine verändernde Evolution fokussiert und für CBCPSe eine erhaltende Evolution. Weiterhin werden durch die Domänenmodelle die unterliegenden Annahmen des technischen Prozesses für die konzipierte Systemarchitektur definiert.

Charakterisierung für CPPSe

CPPSe sind noch stärker als andere CPSe durch eine starke Verzahnung von Physik und Software gekennzeichnet. Deshalb wird darauf im Domänenmodell für CPPSe, das in Abbildung 4.2 gezeigt wird, besonderer Wert gelegt. Das Modell stellt eine mögliche Sichtweise auf CPPSe dar. Diese fokussiert Fertigungsanlagen, da die Evaluation diese als Fallstudie verwendet.

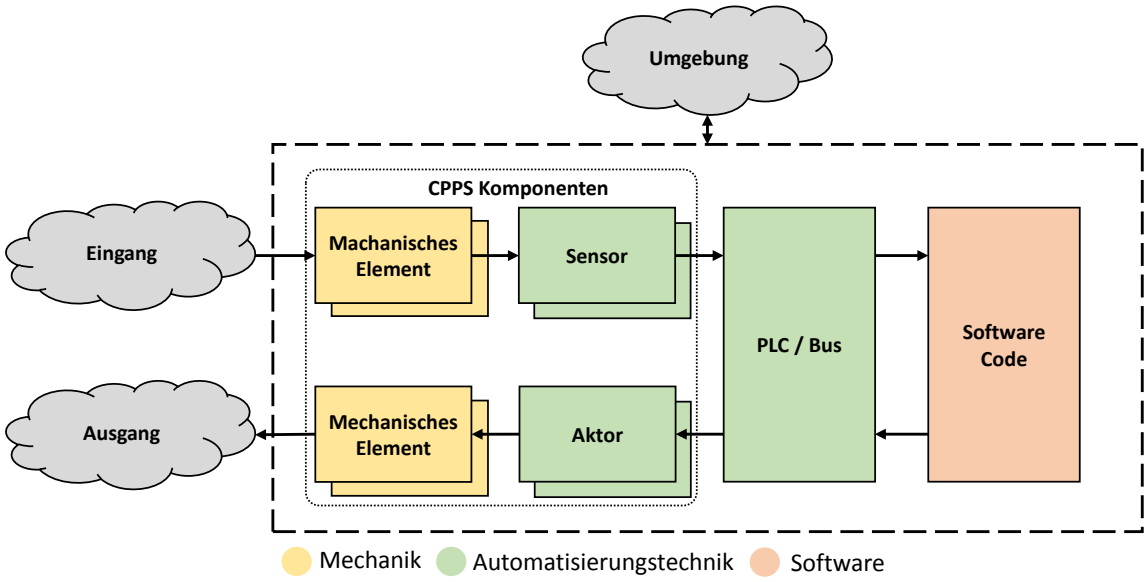


Abbildung 4.2: Domänenmodell für cyber-physische Produktionssysteme

Dem gewählten Modell nach produzieren CPPSe aus einem *Eingang*, wie einem Produktrohling, das fertige Produkt als *Ausgang*. *Mechanische Elemente*, wie bestimmte Werkzeuge, interagieren mit dem Eingang. Diese mechanischen Elemente sind über *Sensoren* und *Aktoren* miteinander verbunden, die über ein *Bussystem* und eine *speicherprogrammierbare Steuerung* automatisiert sind. Dabei wird unter der Verwendung von binären Signalen der Sensoren und Aktoren von einer zyklischen Steuerung in Echtzeit ausgegangen. Die Arbeit nimmt an, dass die betrachteten CPPSe als ereignisdiskrete Systeme modelliert werden können. Dabei ist ein *Ereignis* als der Wechsel zwischen den Elementarzuständen von Signalen definiert und das Verhalten wird als eine Reihe von *Zuständen* beschrieben.

Sowohl die mechanischen Elemente als auch Sensoren und Aktoren sind in *CPPS-Komponenten* organisiert, die jeweils eine spezifische Aufgabe im CPPS erfüllen. Das Verhalten ist durch den Softwarecode vorgegeben, welcher in domänenspezifischen Sprachen geschrieben ist. Da die betrachtete Evolutionsunterstützung insbesondere auch mit dem CPPS verbundene Systeme betrachtet, modelliert das Domänenmodell weiterhin eine *Umgebung*. Diese Umgebung besteht aus anderen interagierenden Systemen, dem Nutzer sowie der elektrischen, mechanischen und softwaretechnischen Einbettung des CPPSs. Es wird angenommen, dass der Steueralgorithmus und damit das CPPS deterministisch agieren, auch wenn dies für die

Umgebungsfaktoren, wie beispielsweise Auftragseingänge durch andere verbundene Systeme, nicht zwangsläufig gilt [SG08].

Im Folgenden wird dieses CPPS-Domänenmodell verwendet, um Kategorien des CRI-Modells für CPPSe abzuleiten. Die Kategorien werden als Kontext der Unterstützungsmethodik aus Kapitel 3 verwendet, welche die automatisiert abgeleiteten Information ergänzen können. Die Kategorien sind nicht disjunkt, weshalb Evolutionsschritte auch in mehrere charakterisierende bzw. örtliche Dimensionen fallen können.

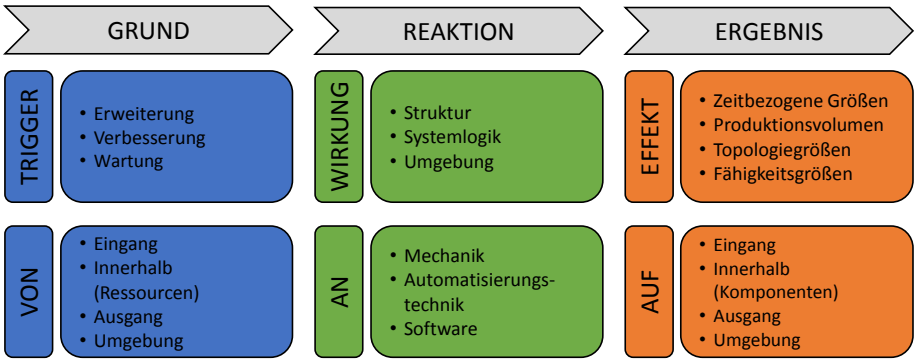


Abbildung 4.3: CRI-Modell für cyber-physische Produktionssysteme

Abbildung 4.3 zeigt das mit den Kategorien vervollständigte CRI-Modell für CPPSe. Diese wurden für alle drei charakterisierende Dimensionen (Grund, Reaktion, Ergebnis) und für die dazugehörigen örtlichen Dimensionen (Von, An, Auf) aus dem Domänenmodell abgeleitet:

Grund Trigger:

Erweiterung: Diese Evolutionsschritte entstehen als Änderungen des Verhaltens, damit angepasste Anforderungen der CPPS-Komponenten erfüllt werden können. Beispielsweise muss ein Produkt mit neuer Beschriftung hergestellt werden, was eine neue Maschine oder Maschinenkonfiguration erfordert.

Verbesserung: Durch den Evolutionsschritt werden die identischen Anforderungen und Ziele der CPPS-Komponenten nun besser erfüllt. Als Beispiel kann die Produktivität bei der Herstellung der Ausgabe verbessert worden sein indem eine Strukturveränderung der Hardwaretopologie vorgenommen wurde.

Wartung: Wartung umfasst jede Veränderung des Verhaltens, die aus einer Aktivität der Wartung resultiert. Dies wäre beispielsweise der Austausch eines alten Sensors, wodurch ein verändertes Verhalten auftritt.

Grund Von:

Eingang: Diese Änderung des Verhaltens rührt von dem Eingangsprodukt selbst her. Beispielsweise davon, dass dieses nach der Evolution schwerer geworden ist.

Innerhalb: Der Grund dieses Evolutionsschrittes stammt aus dem betrachteten CPPS selbst. Dies kann auch nur einzelne CPPS-Komponenten, wie z.B. eine bestimmte Maschine, betreffen. Ein Beispiel wäre eine Erweiterung des CPPSs.

Ausgang: Die Ausgabe des Produktionsprozesses des CPPSs ist der Grund für diese Art der Evolution. Zum Beispiel könnte das fertige Produkt ein zusätzliches Bohrloch benötigen.

Umgebung: Hierbei ist ein Evolutionsschritt aus seiner Umgebung heraus begründet. Beispielsweise kommt dies vor, wenn die Schnittstelle zu einem weiteren System oder dem Nutzer verändert wurde und dies in einer Verhaltensänderung resultiert.

Reaktion Wirkung:

Struktur: Eine solche Reaktion bezieht sich auf eine veränderte Struktur in der Hardware des CPPSs. Es könnte aufgrund eines erhöhten Produktionsvolumen erforderlich gewesen sein, dass die Kapazität erhöht werden musste.

Systemlogik: Statt einer Strukturänderung kann auch die Systemlogik durch einen Evolutionsschritt betroffen sein. Das CPPS agiert nach der Evolution logisch anders, beispielsweise da die Route durch das CPPS verändert wurde.

Umgebung: Die Evolution hat eine Reaktion an der Umgebung erfordert, welche sich auf das CPPS durchschlägt. Beispielweise könnte ein unterschiedlicher Hersteller für eine CPPS-Komponente eingesetzt werden, wodurch zwar die gleiche Funktionalität erzielt wird aber unterschiedliches Verhalten daraus resultieren.

Reaktion An:

Mechanik: Die Reaktion kann hierbei an verschiedenen mechanischen Elementen auftreten, wie beispielsweise an einem bestimmten Bauteil.

Automatisierungstechnik: Eine Reaktion dieser Kategorie bedeutet, dass die Automatisierungstechnik durch den Evolutionsschritt verändert wurde. Ein Beispiel dafür wäre etwa der Austausch eines Sensors.

Software: Die Evolution wird hierbei durch eine Veränderung des Softwarecodes erwirkt, wie beispielsweise bei der logischen Anpassung des Produktionsprozesses.

Ergebnis Effekt: [2]

Zeitbezogene Größen: Der Effekt hat hier einen Einfluss auf die zeitlichen Eigenschaften des CPPSs. Dies könnte z.B. eine Verringerung oder Erhöhung der Produktionsdauer sein.

Produktionsvolumen: Die Evolution verändert die Quantität der im CPPS produzierten Produkte. Dies könnte z.B. das Produktionsvolumen der Ausgabe sein.

Topologiegrößen: Die strukturellen Eigenschaften des CPPSs werden hierbei durch die Evolution verändert - beispielsweise durch eine veränderte Routenführung.

Fähigkeitsgrößen: Jeder Evolutionsschritt dieser Kategorie verändert den Funktionsumfang. Dadurch kann das CPPS z.B. ein verändertes Portfolio an produzierten Ausgaben aufweisen.

Ergebnis Auf:

Eingang: Das Ergebnis wirkt sich auf eine Eigenschaft des Ausgangsprodukts aus. Zum Beispiel kann nur noch eine reduzierte Anzahl von Produkten parallel produziert werden.

Innerhalb (CPPS-Komponenten): In diese Kategorie fallen die meisten Ergebnis, denn diese beschreibt eine Evolution, die sich auf das CPPS selbst oder eine seiner Komponenten auswirkt. Diese könnte z.B. darin bestehen, dass sich die Eigenschaften der strukturellen Beschaffenheit einer Komponente verändert haben.

Ausgang: Dies umfasst jeden Effekt, der sich auf das fertige Ausgangsprodukt auswirkt. Dies könnte z.B. sein, dass ein Produkt zusätzlich beschriftet wird und damit eine zusätzliche Funktionalität besitzt.

Umgebung: Auch die Umgebung kann in ihren Eigenschaften verändert werden. Zum Beispiel so, dass sich die Zuführung von fertigen Produkten an nachgelagerte Systeme verändert.

[2] Die Ergebniskategorien fokussieren, wie im methodischen Kapitel der Arbeit, nichtfunktionale Eigenschaften. Die genutzten Kategorien sind eine leicht angepasste Variante der, im Rahmen des Forschungskontext entwickelten, Kategorien nach Ladiges et al. [LFHL13, Lad18].

Charakterisierung für CBCPSe

Um eine hohe Verfügbarkeit und eine geringe Latenz für eine Vielzahl von Nutzern zu erreichen, speichern und verarbeiten CBCPSe Informationen geografisch über den Globus verteilt [HGH+15]. Das dafür verwendete Domänenmodell legt einen besonderen Fokus auf die Verarbeitung von Informationen und die Kopplung zu vor- und nachverarbeitenden Systemen. Abbildung 4.4 zeigt das Domänenmodell für CBCPSe.

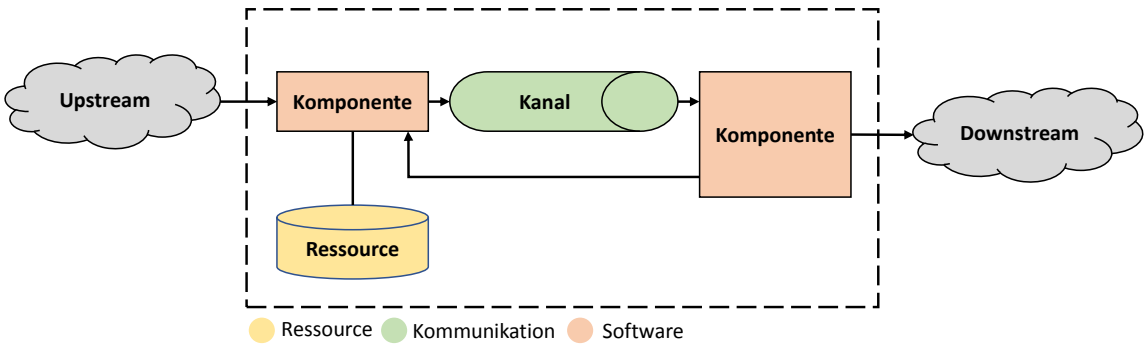


Abbildung 4.4: Domänenmodell für cloud-basierte cyber-physische Systeme

CBCPSe bestehen im Domänenmodell aus drei Typen von Elementen: Der erste Typ sind die Softwarekomponenten, welche die Applikationslogik realisieren. Als zweiter Typ von Elementen werden über Kommunikationskanäle Informationen zwischen den Komponenten transportiert. Und als dritten Typ gibt es Ressourcen, die durch Komponenten genutzt werden können. Für CBCPSe sind dies insbesondere Speicherressourcen.

Das Modell beschreibt eine für Cloud-Systeme typische Weitergabe von Informationen. Der *Upstream* gibt Informationen an Komponenten des CBCPSs ab. Dies können beispielsweise Cloud-Clients sein, die Aufrufe an die Cloud-Infrastruktur tätigen. Diese Informationen werden zwischen Komponenten über *Kanäle* ausgetauscht oder durch Ressourcen genutzt. Nach oder zur weiteren Verarbeitung werden diese Informationen dann an den *Downstream* weitergehen.

Die Kategorisierung für CBCPSe fokussiert *verändernde Evolution*. Eine Übersicht über die entsprechenden Kategorien, welche nachfolgend erläutert werden, zeigt Abbildung 4.5:

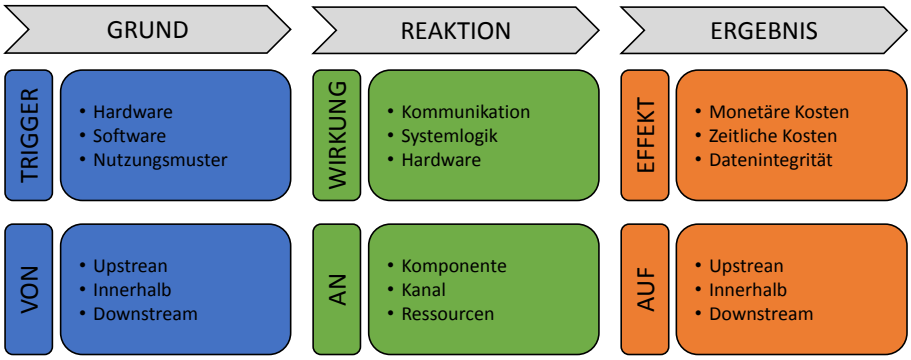


Abbildung 4.5: CRI-Modell für cloud-basierte cyber-physische Systeme

Grund Trigger: [3]

Softwareänderungen: Hierzu gehört jeglicher Grund für eine Evolution, der aus der Applikationslogik kommt. Dies könnte beispielsweise ein durch den Entwickler neu eingespielter Code sein.

Hardwareänderungen: Die Evolution entsteht hier durch eine Änderung der unterliegenden Hardware. Dies könnte z.B. eine ungeplante Änderung beim Ausfall durch korrupte Festplatten oder eine geplante Änderungen bei der Skalierung von Komponenten sein.

Änderung des Nutzungsverhaltens: Die Evolution ist hier in einem veränderten Anfrageverhalten an das CBCPS begründet, wie z. B. ungewöhnlich viele Aufrufe durch den Upstream.

Grund Von:

Upstream: Der Grund für diese Veränderung bezieht sich auf Änderungen der Cloud-Clients oder anderer vorgelagerter Systeme des CBCPSs. Beispielsweise wäre dies eine veränderte Nutzung eines angebotenen Dienstes.

Innerhalb: Der Grund für diese Art der Änderung liegt im CBCPS selbst. Dies könnte z.B. die Aktualisierung von Routingtabellen sein.

Downstream: Hier ist der Evolutionsschritt aufgrund von Abhängigkeiten zu nachgelagerten Systemen entstanden. Zum Beispiel könnte das durch eine Änderung in den Schnittstellen eines Cloud-Providers entstehen.

Reaktion Wirkung:

Ressourcen: Die Reaktion ist hier eine Reduktion oder Erweiterung der Ressourcen. Diese kann gewollt oder auch ungewollt sein. Beispielsweise könnten die Speicher- oder CPU-Kapazitäten beschränkt sein.

Kommunikation: Eine solche Reaktion ist eine Änderung der Erreichbarkeit zwischen den Komponenten. Beispielsweise passiert das, wenn der Kommunikationskanal zwischen zwei Komponente unterbrochen wurde.

Systemlogik: Hierbei entsteht eine Verhaltensveränderung, die den eigentlichen Anwendungszweck des CBCPSs umfasst. Dies bedeutet, dass der Informationsfluss innerhalb des CBCPSs korrekt abläuft, allerdings die Informationen in einer Komponente anders verarbeitet werden.

Reaktion An:

Komponenten: Die Reaktion findet an einer bestimmten Komponente statt. Beispielsweise ist die Verarbeitung in einer Komponente geändert worden.

Kanäle: Der Fluss von Informationen ist an einem bestimmten Kanal gestört. Dies können z.B. ein Verlust oder eine Verzögerung von Ereignissen sein.

Ressourcen: Die Reaktion bezieht sich hier auf eine Ressource, wie z.B. eine Datenbank oder einen Cache-Speicher.

[3] Durch den Fokus auf verändernde Evolution von CBCPSen werden, entgegen des vorgestellten CPPS-Modells, nur Verhaltensveränderungen betrachtet

Ergebnis Effekt: [4]

Monetäre Kosten: Die Evolution führt hier zu erhöhten monetären Kosten. Dies kann beispielsweise aus einer Verletzung von Service-Level-Agreements resultieren oder durch das notwendige Upscaling von Ressourcen verursacht werden.

Zeitliche Kosten: Evolution kann als Ergebnis einen Effekt auf zeitlichen Eigenschaften des CBCPSs haben. Oft ist dies der Fall, wenn durch Änderungen unbefriedigende Antwortzeiten des CBCPSs auftreten.

Datenintegrität: In dieser Kategorie werden alle Ergebnisse eines Evolutionsschrittes erfasst, die in einer erhöhten Angriffsanfälligkeit des CBCPSs resultieren.

Ergebnis Auf:

Upstream: Der Effekt wirkt sich auf vorgelagerte Systeme aus indem beispielsweise erhöhte Fehlerraten bei Cloud-Clients auftreten.

Innerhalb: In diesen Evolutionsschritten treten Ergebnisse direkt im CBCPS auf - beispielsweise indem Komponenten eine verschlechterte Performanz aufweisen.

Downstream: Nachgelagerte Systeme werden durch den Evolutionsschritt beeinflusst. Zum Beispiel werden vermehrt Anfragen an den Downstream gestellt.

Zusammenfassung der Charakterisierung und der funktionalen Ebenen

In diesem Abschnitt wurden funktionale Ebenen eines CPSs definiert. Diese beschreiben, welche grundsätzlichen Arten von Komponenten in einem CPS vorkommen und welche Funktionalität sie abbilden. Für die Evolution wurde festgestellt, dass für eine Evolutionsunterstützung Erweiterungen über mehrere funktionalen Ebenen hinweg notwendig sind. Die daraus resultierenden drei Informationsflüsse sollen durch eine wissenstragende Komponente realisiert werden.

Mit den Klassifizierungen des CRI-Modells wurde zudem gezeigt, dass Evolutionsschritte auch auf spezifische CPSe zugeschnitten werden können. Dafür werden generische Domänenmodelle verwendet. Diese ermöglichen es dem Nutzer Evolutionsschritte zu kategorisieren und dadurch Änderungen und Kontexte zu beschreiben. Das Vorgehen wurde auf die zwei Anwendungsgebiete von CPPSe und CBCPSe angewendet.

Zusammenfassend kann die erste Hypothese bestätigt werden, indem Evolutionsschritte durch domänenspezifische Beschreibungen auf CPSe angepasst werden und in einer wissenstragenden Komponente über die funktionalen Ebenen zu realisieren sind.

[4] Das Ergebnis kann bei den betrachteten Fällen sehr unterschiedlich sein und z.B. durch verschiedene nichtfunktionale Eigenschaften abgebildet werden. Diese Kategorisierung fokussiert verändernde Evolution mit negativem Einfluss, weshalb hier die drei genannten Kategorien verwendet werden. Diese wurden aus verfügbaren Reports großer CBCPSe abgeleitet (siehe [HPR+18, RPH+17]).

4.3 Wissenstragende Komponente für ein CPS

Operativ sind CPSe heterogen in ihrer Struktur und ihrem Verhalten [SKK+12]. Für die wissenstragende Komponente wird dabei zwischen zwei Typen von Softwarekomponenten unterschiedenen: Einerseits sind dies stark in das Hardwaresystem eingebettete Softwarekomponenten, die häufig zyklisch unter Echtzeitbedingungen arbeiten und spezifische Programmiersprachen besitzen. Diese Softwarekomponenten steuern die Hardwareinteraktion über Aktoren und Sensoren. Und anderseits Komponenten in softwaretechnischen Hochsprachen, wie eine objektorientierte Sprache, welche die Funktionen des Bewusstseins und Selbstvergleichs realisieren.

Dabei sollten die Komponenten für ein kompositionales Architekturdesign untereinander homogen kommunizieren können [SKK+12]. Allerdings ist es in einem CPS für die überliegenden funktionalen Ebenen des Bewusstseins und des Selbstvergleichs problematisch oder sogar unmöglich, die Komponenten des technischen Prozesses direkt aufzurufen. Denn eine Integration von Softwarecode in diesen Komponenten, wie es für viele Ansätze der Beobachtung von klassischen Softwaresystemen üblich ist [Rob06], ist technisch aufwändig und durch den Betreiber eines physischen Prozesses oft nicht gewünscht. Entsprechend verfolgt die Arbeit einen nichtinvasiven Ansatz über die integrative Verwendung von Komponenten, Agenten und Diensten.

Für ein kompositionales Architekturdesign sind dabei Funktionalitäten in abgeschlossen Softwareeinheiten mit klaren Abhängigkeiten zu kapseln, die dann über flexible und in gleicher Weise wiederverwendbare Einheiten eine homogene Kommunikation ermöglichen. Komponenten ermöglicht eine solche Unterscheidung von logisch getrennten Funktionalitäten. Und eine dienstorientierte Sichtweise ermöglicht eine hohe Effizienz und Agilität in der Kommunikation zwischen interoperablen Funktionalitäten sowie die Möglichkeit der Erfüllung übergeordneter Zielstellungen [Erl09]. Smarte Funktionalitäten der übergeordneten funktionalen Ebenen benötigen weiterhin ein hohes Maß an Autonomie und automatisierter Verwaltung, welche vorrangig in der (Multi-)Agentenforschung verfolgt wird. Entsprechend wird hier auch der bereits im Grundlagenkapitel erläuterte Ansatz der *Aktiven Komponenten* verwendet, der dienstorientierte Komponenten mit Agentenarchitekturen verbindet (siehe [PB13]).

Die damit für die Realisierung von CPSen **entwickelte Systemarchitektur** ist in Abbildung 4.6 dargestellt. Diese Architektur wird nachfolgend sukzessive vom technischen Prozess ausgehend aufgebaut und erläutert. Dabei wird zunächst beschrieben, wie Quellartefakte in Funktionalitäten von Komponenten softwaretechnisch abgebildet werden. Weiterhin werden diese über eine mit Diensten verknüpfte hierarchische Komponentenstruktur als Informationsartefakte zusammengeführt und konsistent gehalten (Abschnitt 4.4).

Diese Artefakte werden dann auf der Ebene der cyber-physischen Verbindung verwendet, um Modellartefakte zur Laufzeit durch Dienste und Agenten aktiv zu verwalten und steuern (Abschnitt 5). Abschließend wird gezeigt, wie wissenstragende Komponenten in einem verteilten Marktplatz auch so agieren können, dass einzelne Evolutionsschritte auch als Einheiten für die Verbesserung von ähnlichen Problemlösungen anderer Komponenten in einem verteilten Umfeld realisiert werden können (Abschnitt 4.6).

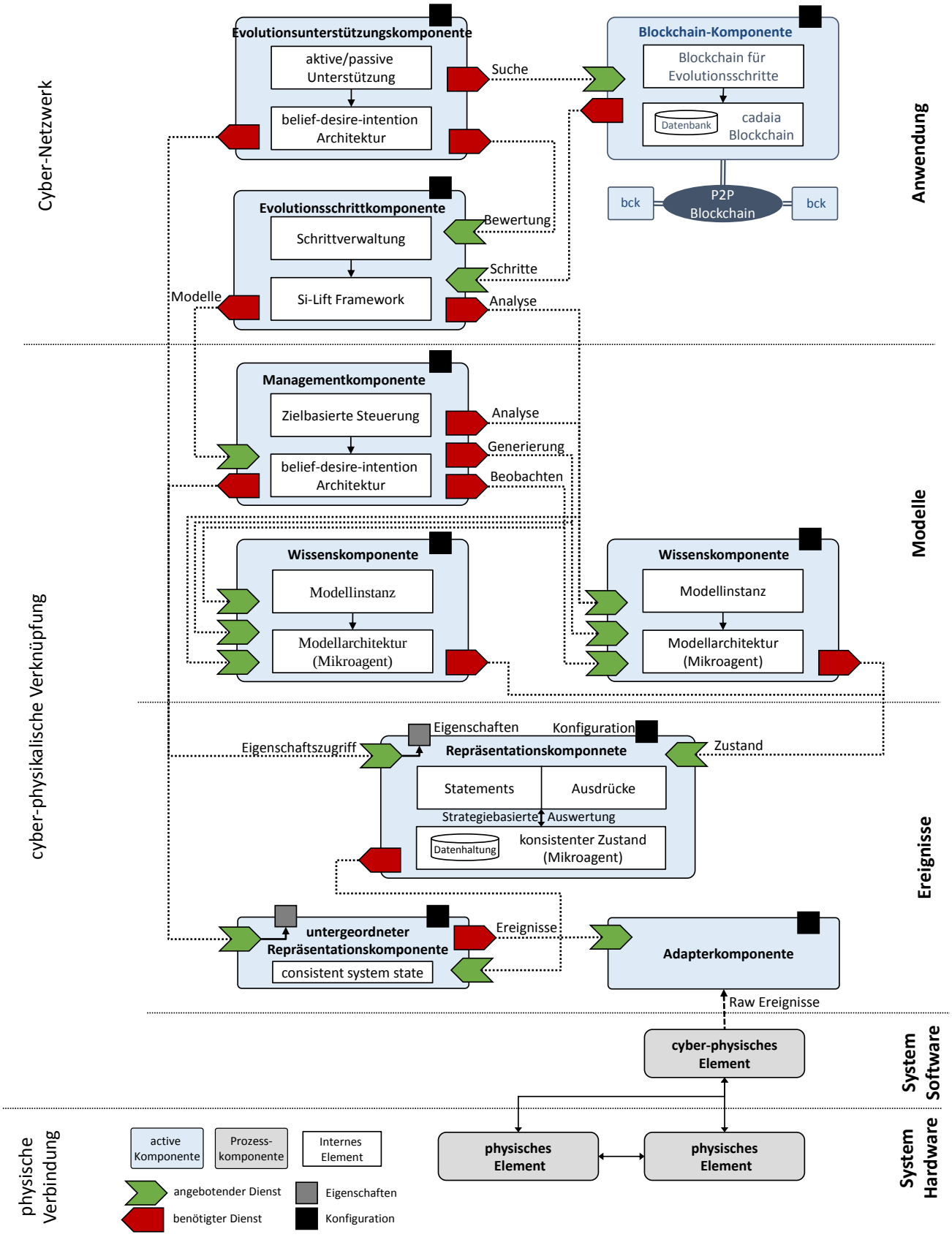


Abbildung 4.6: Aktive Komponentenarchitektur der Evolutionsunterstützung

4.4 Konsistente Fassade für heterogene Ereignis- und Kontextquellen

Für eine wissenstragende Komponente müssen Quell- und Kontextartefakte softwaretechnisch abgebildet werden. Dafür werden mit den Adapter- und Repräsentationskomponenten zwei unterschiedliche Typen von Komponenten verwendet.

4.4.1 Integration von Ereignisquellen in heterogenen cyber-physischen Systemen

In der Systemarchitektur dieser Arbeit dienen **Adapterkomponenten** dem Zweck, die heterogenen Ereignisquellen als Softwarekomponente in die wissenstragende Komponente zu integrieren. Dadurch können Ereignisse der Ausführung oder der Entwicklung erfasst werden. Die Adapterkomponenten schaffen - dem Proxy-Pattern folgend - für jedes betrachtete Quellartefakt einen transparenten Zugang zur Ebene der cyber-physischen Verbindung. Zusätzlich kapseln sie die Ereigniserzeugung und -beobachtung gegenüber den Komponenten der oberen CPS-Ebenen. Beide Funktionalitäten werden über einen Dienst übergeordneten Komponenten bereitgestellt. Die Adapterkomponente besitzt so durch eine eigene Ausführungssteuerung und Verwaltung ihrer Threads einen hohen Grad an Freiheit bei der Akquirierung von Ereignissen, Dadurch kann sie u.a. autonom Verbindungen zum technischen Prozess aufbauen und diese halten. Dies ermöglicht auch domänenspezifische und hardwarenahe Technologien, wie z.B. Prozesskontrollprotokollen, einzusetzen.

Als zusätzliche verarbeitende Funktionalität realisieren die Adapterkomponenten spezielle Vor- und Nachverarbeitungsschritte. Diese sind im Ereignisdienst der Komponente realisiert (siehe Abbildung 4.6) Die realisierte Funktionalitäten sind insbesondere das Caching von Ereignissen sowie die Entkopplung der unterschiedlichen Raten, in denen Ereignisse akquiriert und konsumiert werden. Dies bedeutet, dass die Raten, in denen Ereignisse von der Adapterkomponente aufgenommen werden (*source rate*) von der Rate, unter der sich eine nutzende Komponente bei dem Dienst der Adapterkomponente registriert, getrennt werden (*desired rate*). Dies hat den Grund, dass die Anforderungen an die Rate der übergeordneten Funktionalitäten der Evolutionsunterstützung unterschiedlich sind. Dies erfordert, dass Komponenten dieser Funktionalitäten nach Ereignissen unterschiedlicher Quellen suchen müssen und sich dann über ein Publish-Subscribe Mechanismus für eine beliebige Untermenge dieser Ereignisse registrieren können. Dazu wird angenommen, dass der Wert eines Ereignisses immer bis zum nächsten überschreibenden Ereignis dieses Ereignis- oder Kontexttyps gültig bleibt.

Jede Adapterkomponente integriert damit ein Quell- oder ein Kontextquellartefakt, welches jeweils, wie im vorherigen Kapitel gezeigt, ganz unterschiedliche Typen von Ereignissen bereitstellen können. Dabei ist die Implementierung der Adapterkomponenten domänen- und technologiespezifisch wodurch gewährleistet wird, dass die Evolutionsunterstützung Quellartefakte für verschiedene Modell- und Systemtypen verwenden kann. Für die Implementierung in dieser Arbeit wurden folgende Adapterkomponente konkret technisch implementiert:

- Zur technischen Realisierung der Adapterkomponenten für CPPSe wurden die auf einem Bussystem vorkommenden Signale in einer Schnittstelle für die *Open Platform Communications Unified Architecture* (OPC UA) definiert. Damit wurden die Signalveränderungen auf dem Bussystem über ein OPC-Gateway beobachtet. Für Kontextinformationen wurde eine Informationsmodell der Ort- und Zeitkontexte für alle definierten Signale in der Adapterkomponente hinterlegt.

  Die Simulationsumgebung, wie sie methodisch bei Testfällen angewendet wird, nutzt zusätzlich die Laufzeitumgebung von Lochau et al. [LBL+14]. Diese führt dabei Testfälle

aus und erzeugt für alle Testfälle entsprechende OPC-Ereignisse. Diese Ereignisse werden über die Adapterkomponenten für CPPSe aufgenommen.

- Für CBCPSe wurde die Jadex-Plattform für *Aktive Komponenten* verwendet, die eine Beobachtung von Dienstaufrufen ermöglicht. Die an der Dienstschnittstelle getätigten Aufrufe, werden dabei intern durch den Future-Mechanismus verwaltet und beobachtet (siehe Abschnitt 2.3.4). Für die Bildung von Nachrichtensequenzen wurde, wie methodisch bereits erläutert, das Verfahren von Rohr [Roh15] auf die verfügbaren Informationen von Aufrufen der *Aktiven Komponenten* angewendet.

  Die Quellcodeanalyse wird für den Kontext von CBCPSen durch einen entsprechenden Parser analysiert (siehe Abschnitt 3.3.2). Der Parser wurde parallel zur wissenstragenden Komponente ausgeführt und gibt die Kontexte über eine Schnittstelle an die Adapterkomponente ab.

Alle Adapterkomponenten können über die Konfiguration von *Aktive Komponente* für die spezifischen Systeme konfiguriert werden. Dies ist beispielsweise notwendig, wenn unterschiedliche Ausprägungen von CPPSen beobachtet werden, die für ihre verfügbaren Signale eine abweichende Konfiguration benötigen.

Zusammenfassend fangen die Adapterkomponenten aus softwaretechnischer Sicht die Heterogenität von Ereignisquellen in CPSen ab und verringern in der vorgeschlagenen Systemarchitektur die Komplexität der ebenenübergreifenden Interaktionen.

4.4.2 Repräsentation in einer konsistenten Systemfassade

Ein zentraler Vorteil eines objektiven Bewusstseins ist es, dass die wissenstragende Komponente die unterliegende technische Implementierung und Wartungsroutinen nicht ändert oder einschränkt. Softwaretechnisch hat diese Separierung den Vorteil, dass die Evolutionsunterstützung strikt von dem steuernden Code des technischen Prozesses getrennt bleibt und damit von diesem entkoppelt ist. Dennoch wird für die Evolutionsunterstützung ein Bewusstsein über alle funktionalen Ebenen des CPSs hinweg benötigt.

Dafür ist eine *Easy-to-use*- und *Easy-to-deploy*-Architektur für smarte Funktionalitäten im Sinne von Zhang et al. [ZT16] zu implementieren. Um diese zu erreichen, wird über **Repräsentationskomponenten** eine durchgängige Systemfassade zwischen dem technischen Prozess und der Evolutionsunterstützung gebildet. Repräsentationskomponenten können entlang der technischen Komponenten, aber auch organisatorisch entlang von beispielsweise Geschäftsprozessen aufgebaut werden. CPSe sind allerdings in der Regel hierarchisch unterteilt und organisiert [LBK15]. Eine solche Topologie ermöglicht es, ein CPS in verschiedenen Granularitäten zu dekomponieren. Dies ist auch für die Evolutionsunterstützung gewünscht. Entsprechend sollten, wie es Fröhlich und Nejdl [FMNS97] als semantische Verteilung für eine agentenbasierte Diagnostik vorschlagen, die Evolutionsartefakte der Evolutionsunterstützung entlang der technischen Hierarchie verwaltet werden. Hierbei zeigt sich die Notwendigkeit eines komponentenbasierten Vorgehens, durch welches das Wissen in für sich abgeschlossene Komponente mit klaren gegenseitigen Abhängigkeiten repräsentiert wird.

Wie Maga et al. [MJG11] feststellen, ist die geeignete Verknüpfung und Granularität einer solchen Komponentenhierarchie stark kontextabhängig und korreliert mit der angestrebten Wiederverwendung der Komponenten. Entsprechend können, abhängig vom gewünschten Design und Ziel für die Systemarchitektur, unterschiedliche Verknüpfungen und Granularitäten gewählt werden. Dabei ist zu beachten, dass sich die Verknüpfung im Allgemein an den Verantwortlichkeiten im abgebildeten technischen Prozess orientieren sollte, damit diese in der System-

fassade erhalten bleibt. Die Arbeit verfolgt dabei eine 1-zu-1-Verknüpfung zwischen den technischen Komponenten des CPSs und den korrespondierenden Repräsentationskomponenten. Hierfür wird eine Granularität auf Komponentenebene gewählt. Dies entspricht der im industriellen Umfeld verbreiteten Sichtweise eines digitalen Zwillings (*digital twin*), indem die Repräsentationskomponenten ihren jeweiligen technischen Zwilling für die Evolutionsunterstützung in der wissenstragenden Komponente abbilden. Durch diese Verknüpfung ist jede Repräsentationskomponente für jeweils ihren Zwilling zuständig und muss einen synchronisierten und konsistenten Zustand gegenüber diesem autonom erhalten.

Makroskopisches Wissen über mehrere Komponenten hinweg wird über die, in Abbildung 4.6 bereits dargestellte, Hierarchie der Repräsentationskomponenten verwaltet. Dabei werden auf höheren Hierarchieebenen die Ereignisse der niedrigen Repräsentationskomponenten genutzt. Um die Verantwortlichkeiten in der Hierarchie zu erhalten, werden Ereignisse dabei indirekt genutzt. Indirekt bedeutet, dass eine Repräsentationskomponente auf niedrigem Hierarchielevel aggregierte oder gefilterte Ereignisse anbieten kann, die dann von den überliegenden Repräsentationskomponenten konsumiert werden. Durch dieses Vorgehen wird eine makroskopische Sichtweise innerhalb der Systemfassade komponiert und das Evolutionswissen nicht nur horizontal, sondern auch vertikal aufgebaut.

Zusätzlich können Repräsentationskomponenten die inhärenten Eigenschaften ihres jeweiligen technischen Zwillings abbilden, um diese in der wissenstragenden Komponente zu repräsentieren. Dies wird im Rahmen dieser Arbeit genutzt, um die berechneten Eigenschaften der Modellartefakte für andere Komponenten und den Nutzer zugreifbar zu machen.

Wiederverwendbarkeit wird bei Repräsentationskomponenten durch ihre Konfiguration erreicht, indem jede Repräsentationskomponente bezüglich ihrer Zuständigkeit konfiguriert werden kann. Dies geschieht über ein Filtern auf Basis der Kontextinformationen der Ereignisse. Durch dieses Vorgehen kann eine Wiederverwendbarkeit und Flexibilität beim Aufbau der Systemfassade gewährleistet werden.

Erzeugen von konsistenten Zuständen

Für weitere Verarbeitungsmöglichkeiten bei der Evolutionsunterstützung ist ein konsistenter Zustand auf Ebene der cyber-physischen Verbindung entscheidend. Diese Konsistenz an der Systemfassade erhalten Repräsentationskomponente nach dem *BASE-Prinzip (basically available, soft state, eventual consistency)*. Dabei besitzen die Ereignisse bereits eine Ordnung, die durch ihren Zeitkontext gegeben ist (vergleiche Abschnitt 3.3.1). Entsprechend muss die Repräsentationskomponente nur die Ordnung zwischen den verschiedenen verteilten Adapterkomponenten erhalten.

Konsistenz bedeutet in diesem Zusammenhang, dass die Repräsentationskomponenten immer einen korrekten Zustand aller verteilten Adapterkomponenten zu einer bestimmten Zeit (*snapshot*) vorhalten. Für diese Zustandserstellung wird der stream-orientierte Programmierstil des Futurekonzepts von *Aktiven Komponenten* (siehe Abschnitt 2.3.4), der vergleichbar zum *Complex Event Processing* funktioniert, in dieser Arbeit mit einem Snapshot-Verfahren in den Repräsentationskomponenten ergänzt. Unter dem Ausführungskontext wird durch die Adapterkomponente ein Future erstellt, welches die Lieferung von Ereignissen bis zu einem bestimmten Zeitpunkt sicherstellt. Dies umfasst alle Ereignisse, die bis zum Zeitpunkt t auftreten sind und durch das Future an den Ausführungskontext der Repräsentationskomponente übermittelt wurden. Diese Übermittlung kann dabei durch die Repräsentationskomponente verifiziert werden.

Um die Konsistenz über verschiedene Adapterkomponenten herzustellen, wird der Snapshot-Algorithmus nach Coulouris et al. [CDK05] auf diesen Fall angewendet. Die wesentlichen Schritte der Repräsentationskomponente für einen konsistenter Gesamtzustandes sind dabei folgende:

1. Wenn ein neuer Gesamtzustand berechnet werden soll, wird zunächst der aktuelle Gesamtzustand in der Repräsentationskomponente gespeichert. Zusätzlich wird eine Markierungsnachricht über ein Ausgangs-Future an alle Adapterkomponenten gesendet. Ab diesem Zeitpunkt werden für jede Adapterkomponente alle Ereignisse aus dessen Ausgangs-Future im Gesamtzustand berücksichtigt. Dabei besteht genau ein Ausgangs-Future zwischen jeder Adapterkomponente und jeder Repräsentationskomponente.
2. Wenn eine Adapterkomponente eine Markierungsnachricht empfängt, wird diese durch das Ausgangs-Future der entsprechenden Repräsentationskomponente zurückgesendet. Wenn die Repräsentationskomponente diese Markierungsnachricht zurückerhält, werden nachfolgende Ereignisse nicht mehr bei der Aufzeichnung berücksichtigt.

Durch dieses Verfahren ist sichergestellt, dass alle Ereignisse bis zu dem Zeitpunkt der Markierungsnachricht im konsistenten Gesamtzustand berücksichtigt sind. Dabei ist zu beachten, dass dieser Ansatz zwar immer einen konsistenten Gesamtzustand erzeugt, aber keine Echtzeitanforderungen erfüllt. Dadurch kann immer nur der letzte konsistente Zustand der Repräsentationskomponente für die Evolutionsunterstützung genutzt werden. Dies bedeutet, dass womöglich weitere, bereits von Adapterkomponenten übermittelte Ereignisse noch nicht im konsistenten Gesamtzustand berücksichtigt wurden. Der durch das Verfahren erzeugte Gesamtzustand entspricht allerdings dem notwendigen Informationsartefakt der Evolutionsunterstützungsmethode.

Der Gesamtzustand wird softwaretechnisch als ein Datenobjekt in einem Zustandsdienst gehalten (vergleiche Abbildung 4.6), was im folgenden Abschnitt noch näher betrachtet wird. Dadurch bleibt die folgende Wissensverarbeitung von der Beobachtung und Verarbeitung der CPS-Ereignisse unberührt. Dies ermöglicht auch die Einbindung weiterer wissensverarbeitender Methoden in die hier vorgestellte Systemarchitektur.

Die Ereignisse der Repräsentations- und Adapterkomponenten werden verteilt angeboten indem sie über die *Awareness*- und *Discovery*-Funktionalität von *Aktiven Komponenten* gesucht und genutzt werden. Dienstaufrufe sind dabei gegenüber Nebenläufigkeit und korrupten Zuständen geschützt, indem der Zugriff innerhalb ihres Ausführungskontextes linearisiert ist. Weiterhin werden die zur Verfügung stehenden Ereignisse der Evolution unveränderbar (*immutable*) gehalten bzw. automatisiert geklont (vergleiche dazu die den zugrunde liegenden Ansätze nach Pokahr et al. [PB13]).

Zusammenfassung der Integration von Ereignissen

In diesem Abschnitt wurden zwei Systemkomponenten vorgestellt, die die Funktionalitäten der Quell-, Kontext- und Informationsartefakte softwaretechnisch umsetzten können. Die Adapterkomponenten ermöglichen dabei beliebige heterogene Ereignisquellen zu integrieren und über Dienste innerhalb der Systemarchitektur homogen zugreifbar zu machen. Mit den Repräsentationskomponenten wird ein digitaler Zwilling realisiert, der Gesamtzustände konsistent erzeugen und verwalten kann.

Dies ist, der zweiten operativen Hypothese folgend, der ersten Schritt zu einem objektiven Bewusstsein einer Evolution.

4.5 Dokumentation durch dienstorientiertes Wissensmanagement

Da für ein Bewusstsein von Evolution das Verhalten nachvollziehbar abgebildet und aktuell gehalten werden muss [SHEA10], wird der Gesamtzustand der Repräsentationskomponenten genutzt, um die Modelle aus der Evolutionsunterstützung in eine Komponentenarchitektur zu integrieren. Diese Modelle müssen zur Laufzeit über ihren gesamten Lebenszyklus verwaltet werden und sind für ein dienstorientiertes CPSs als notwendig zu erachten [DLV12, HLLF14b].

Die dafür vorgeschlagenen Wissenskomponenten zeigte bereits Abbildung 4.6. Die Systemarchitektur entspricht diesbezüglich den generellen Leitlinien für anforderungsüberwachende Systeme (*requirement monitoring system*) nach Robinson [Rob06]. Eine Modellebene vermittelt dabei nach Robinson zwischen der Ereignisebene, welche Ereignisse akquiriert, (vor)verarbeitet und in einem Zustand speichert, und den eigentlichen Anwendungskomponenten [Rob06]. Die dafür verwendeten, domänenspezifischen Modellartefakte (siehe Abschnitt 3.4) werden softwaretechnisch in Wissenskomponenten gekapselt. Diese werden in Abschnitt 4.5.1 erläutert. Sie sind nach außen über Dienstaufrufe mit den Ereignissen verknüpft und können von weiteren verarbeitenden Komponenten genutzt werden. Dabei wird ein dienstorientierter Ansatz verwendet, da dieser als wesentliche Verbindungstechnologie der Realisierung von CPSen angesehen werden kann [WW18].

Zur Steuerung des Ablaufs des dadurch realisierbaren Bewusstseins wird eine Managementkomponente verwendet (Abschnitt 4.5.2) Diese gewährleistet einen automatisierten Erhalt der verschiedenen Wissenskomponenten durch eine dienstbasierte Steuerung der Koevolution.

4.5.1 Wissenskomponenten zur Laufzeit

Die Verarbeitung von Verhaltensmodellen ist modular jeweils in Komponenten separiert. Denn Verhaltensmodelle für CPSe sind in der Regel heterogen und komplex [DLV12], weshalb es schwierig ist, eine einheitliche Systemarchitektur für die Integration zu finden. Um diesem Problem zu begegnen, werden für Wissenskomponenten die unterschiedlichen internen Architekturen von *Aktiven Komponenten* genutzt, welche jeweils ihre eigene Ausführungsumgebung besitzen und allein über Dienstaufrufe verändert werden können

Diese **Wissenskomponenten** können auf jeder Hierarchiestufe der Repräsentationskomponente angebunden werden und dienen der Beschreibung des dem CPS inhärente Wissen. Die Wissenskomponenten sind somit die softwaretechnische Implementierung von Modellartefakten. Die Integration in die Systemfassade wird durch die Nutzung des Zustandsdienstes der Repräsentationskomponenten realisiert.

Dafür bietet der Zustandsdienst ein in dieser Arbeit entwickeltes Verfahren von strategieorientiert ausgewerteten Aussagen für Zustandszugriffe an (vergleiche Abbildung 4.6). Das Grundprinzip ist dabei, dass die Wissenskomponente eine **Aussage** über einen Dienstaufruf an die Repräsentationskomponente weiterleitet. Diese werten die Aussage anhand ihres konsistenten Zustands wiederkehrend aus und informieren die Wissenskomponente, wenn die in der Aussage formulierte Situation eintritt. Die Aussagen bestehen aus drei wesentlichen Teilen:

1. Jede *Aussage* besitzt einen *Ausdruck* über den Zustand der Repräsentationskomponente. Dieser Ausdruck beschreibt, zu welchen Ereignistypen die Aussage gemacht wird. Entsprechend können beliebige Ereignistypen miteinander kombiniert werden. Das Resultat einer Auswertung eines Ausdrucks zu einem bestimmten Zeitpunkt wird als *Ausdruckswert* bezeichnet. Der Ausdruckswert wird innerhalb der Ausführungsumgebung der Repräsentationskomponente und nicht der Wissenskomponente ausgewertet, um eine konsistente Auswertung innerhalb der Systemfassade zu gewährleisten.

2. Neben dem Ausdruck besitzt jede Aussage auch einen zum Ausdruck passenden *Matcher*. Dieser Matcher vergleicht zu einem beliebigen Zeitpunkt den momentanen Ausdruckswert einer Aussage mit einem gewünschten Wert. Wenn der gewünschte Wert zu einem Zeitpunkt mit dem Ausdruckswert übereinstimmt, wird ein Ereignis generiert. Dieses wird über einen Callback-Mechanismus an die Wissenskomponente übertragen. Dies geschieht mit einem Dienstaufruf über ein wiederverwendbares Future, dass bei der Übergabe der Aussage als Rückgabewert hinterlegt wird. Dabei wird bei jeder positiven Prüfung ein Ereignis generiert, sodass ein stetigen Ereignisfluss, wie er methodisch zwischen Informationsartefakt und Modellartefakt als Folge modelliert wurde, entstehen kann.
3. Da die Auswertung der Aussage in der Repräsentationskomponente gekapselt ist und die Zeit zu der eine Aussage erfüllt wird unbekannt ist, verfügen Aussagen als dritten Teil über eine *Auswertungsstrategie*. Diese ermöglicht eine effizientere Auswertung einer Vielzahl von Aussagen. Denn dadurch werden Aussagen nicht zu jedem möglichen Zeitpunkt ausgewertet, sondern die auszuwerten Zeitpunkte werden durch die Strategie in Intervallen vorgegeben. Die Repräsentationskomponente fragt dazu das nächste Intervall der Auswertung bei der Strategie ab und eine entsprechende Auswertung wird daraufhin im Scheduler seiner Ausführungsumgebung hinterlegt. Wenn die Auswertung zur Ausführung eingeplant (*schedule*) ist, wird die Aussage auf dem dann aktuellen Zustand ausgewertet. Das Intervall kann im implementierten Prototyp der Evaluation linear, zunehmend oder abnehmend zu einer bestimmten Deadline verlaufen.

Zur Erläuterung der Grundstruktur zeigt der Quellcode 4.1 einen Ausschnitt aus der unterliegenden Klasse einer Aussage (*StateCondition¡T¿*). Dabei wird die Aussage mit einem Ausdruck (*IExpression¡T¿*), einem Matcher (*Matcher¡? super T¿*) und einer Strategie (*IEvaluationStrategy*) initialisiert. Daraufhin kann die *await*-Methode vom Zustandsdienst genutzt werden, um der Strategie folgend auf ein positives Ergebnis des Matchers bezüglich des Ausdrucks zu warten.

```
1  @Reference(local = true, remote = false)
2  public class StateCondition<T>{
3    @Include protected final IExpression<T> expression;
4    @Include protected Matcher<? super T> matcher;
5    @Include protected IEvaluationStrategy evaluation;
6
7  public StateCondition(final IExpression<T> expression, final Matcher<?
       super T> matcher, final IEvaluationStrategy evaluation)
8  {
9  ...
10 }
11
12 public T await(IEvaluationFunction<? super IExpression<T>, T> function,
       final long timeout, final long starttime) throws TimeoutException
13 {
14   return evaluation.until(expression, function, matcher, timeout,
         starttime);
15 }}
```

Quellcode 4.1: Basisklasse für Aussagen in Java

Das konkrete Zusammenspiel zwischen den verschiedenen Teilen zeigt Abbildung 4.7 beispielhaft. Auf der linken Seite wird eine Aussage über den Zustand definiert, indem die *Summe* von zwei Ereignisse geprüft wird. Und auf der rechten *Seite* wird eine Aussage formuliert, die eine logische UND-Verknüpfung vornimmt. Die Ausdrücke sind softwaretechnisch nicht beschränkt, da diese als Objekte, die ein generisches Interface erfüllen, definiert werden. Die Aussagen bestehen aus einem Matcher, der die Aussagen auf das gewünschte Ergebnis (2 bzw. $wahr$) überprüft. Dieser Matcher kann ebenfalls eine beliebige Implementierung aufweisen, sofern er entsprechende Interfaces implementiert.[5] Zusätzlich wird die Aussage über die Ausdrücke ($e_1 + e_2$ bzw. $e_1 \wedge \neg e_2$) spezifiziert.

Der Zustand des CPSs wird dabei durch die Repräsentationskomponente eingebunden. Entsprechend wird der Ausdruck unter diesem Zustand ausgewertet. Beispielsweise könnte $e_1 + e_2 = 1$ oder $e_1 \wedge \neg e_2 = wahr$ im momentanen Zustand gelten. Wann eine Auswertung stattfindet, wird durch die gegebene Strategie bestimmt. Im Beispiel ist in beiden Fällen eine abnehmende Strategie mit einer Deadline von 1000 ms gegeben. Abnehmend bedeutet, dass die Ausdrücke am Anfang jedes Intervalls nur seltener geprüft werden und bei nahender Deadline entsprechend häufiger. Nach jeder Auswertung überprüft der Matcher, ob die ausgewerteten Ausdrücke dem gewünschten Wert entsprechen. Im Beispiel wäre dies für die Verknüpfung der Fall ($wahr = wahr$) und für die Summe nicht der Fall ($1 \neq 2$). Wenn der Matcher eine positive Prüfung vorgenommen hat, wird die Wissenskomponente, die diese Aussage hinterlegt hat, informiert.

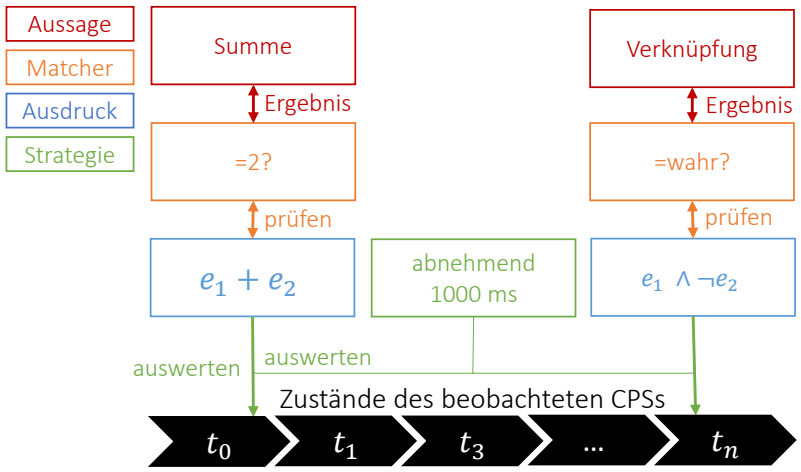


Abbildung 4.7: Beispielhaftes Vorgehen für strategieorientiert ausgewerteten Aussagen

Zusammenfassend vermitteln Wissenskomponente für ihre verwalteten Artefakte einen Pool von Aussagen und übertragen diesen als Ausdrücke auf den Zustand des CPSs für eine strategieorientierte Auswertung an Repräsentationskomponenten. Der entwickelte Ansatz von Aussagen über einen Zustandsdienst kann auch in anderen ereignisorientierten Bereichen als Alternative zum häufig verwendeten Pull-Prinzip eingesetzt werden. Beispielsweise können Testfälle von Ereignissen, bei den der Eintrittszeitpunkt unbekannt ist, durch diesen Ansatz effizienter ausgewertet werden (vergleiche [HLF18]).

[5] In der prototypischen Implementierung wurde das Framework Hamcrest verwendet (http://hamcrest.org/).

4.5.2 Steuerung der Evolutionsunterstützung

Mit den durch die Aussagen gewonnen Informationen können Wissenskomponenten ihre jeweiligen Modelle anpassen und verbessern. Eine solche Konfiguration von *Model@runtime*-Artefakten umfasst typischerweise Funktionalitäten wie Entscheiden, Analysieren und Lernen [AGJ+14]. Für die Evolutionsunterstützung müssen die in Abschnitt 3.4 definierten Funktionalitäten der Beobachtung, der Generierung und der Analyse umgesetzt werden. In der Systemarchitektur verarbeiten Wissenskomponenten die Ereignisse deshalb in unterschiedlichen Modi, welche in jeweils einem Dienst umgesetzt sind (vergleiche Abbildung 4.6). Dieser Dienst stellt eine Anweisung dar, wie das Modell der Wissenskomponente durch ein Ereignis beeinflusst wird.

Die so umgesetzten Dienste sind durch die Systemarchitektur wie folgt implementiert:

1. Der Dienst der Generierung ermöglicht es, ein Modell aus Ereignissen von Aussagen neu zu konstruieren. Als Resultat entsteht ein neues oder veränderten Modell. Um die Generierungsverfahren des methodischen Kapitels zu implementieren, wird der *Micro-Kernel* von *Aktiven Komponenten* genutzt. Für die Generierungsverfahren wird das *Eclipse Modeling Framework (EMF)*[6] verwendet. Dieses kann dynamische Modelle basierend auf *Ecore*-Meta-Modellen erzeugen. Eine Modellinstanz dieser Meta-Modells wird dann, den jeweiligen Generierungsverfahren folgend, programmatisch in Java-Code erzeugt.
2. Der Dienst der Beobachtung ermöglicht es, ein vorhandenes Modell mit aktuellen beobachten Werten abzugleichen. Dadurch können durch Abweichungen vom Modell konträre Aussagen erkannt werden. Dazu wird die Instanz des EMF-Modells geladen und im Dienst der Beobachtung mit den aktuellen Ereignissen über in einer Repräsentationskomponente hinterlegten *Aussagen* verglichen. Für CPPSe werden dabei Token in der dynamischen EMF-Instanz des Modells durch die Ereignisse gefeuert, sodass diese durch die entsprechenden Petri-Netze wandern. In CBCPS-Modellen wird die EMF-Instanz mit den jeweiligen Ereignissen abgeglichen und verifiziert. Dies inkludiert den Abgleich der wahrgenommenen Komponenten, Dienste und Methoden.
3. Der Dienst der Analyse nimmt ein vorhandenes Modell und bestimmt die methodisch definierten Eigenschaften. Dazu werden zunächst die EMF-Instanzen dynamisch geladen und dann das jeweilige Analyseverfahren durchgeführt. Nach der Analyse werden die Werte der Eigenschaften als *JavaBeans* an die Repräsentationskomponente weitergereicht, damit auf diese durch andere Komponenten zugegriffen werden kann. Dazu existiert ein Dienst der Eigenschaftszugriffs an den Repräsentationskomponenten (vergleiche Abbildung 4.6).
4. Für CPPSe wurde bei Materialflussmodellen im Rahmen dieser Arbeit zusätzlich mit einer Simulation des Modells experimentiert, um auch eine Prognose von Eigenschaften erstellen zu können. Diese Simulation nutzt eine ereignisorientierte Simulationsumgebung, welche fiktive Aussagewerte erzeugt, wodurch das Verhalten des Modells in simulierten Umgebungen untersucht werden kann. Dabei wurde dieser Ansatz funktional bestätigt, allerdings in dieser Arbeit danach nicht weiterverfolgt.

Mit den so geschaffenen Diensten kann eine Selbstdokumentation erreicht werden. Diese folgt grundsätzlich der *MAPE-K*-Schleife für selbstwahrnehmende Systeme, die das Beobachten, die Analyse, das Planen und die Ausführung vorsieht [ARS15]. Dabei sind Beobachtung und Analyse in der Systemarchitektur analog abgebildet. Die Ausführung entspricht in der Evolutionsunterstützung der Generierung, also der Wissensgewinnung. Eine Planung wird

[6] https://www.eclipse.org/modeling/emf/

entsprechend der identifizierten Rahmenbedingungen nur manuell durchgeführt, da eine grundlegende Annahme dieser Arbeit ist, dass die Entscheidung darüber jeweils beim Nutzer belassen wird. Entsprechend handelt es sich um eine halbautomatisierte Dokumentation, wobei die Systemarchitektur eine automatisierte Entscheidung, beispielsweise durch Agentenarchitekturen in einer weitere Entscheidungskomponente, zulässt. Dieser Aspekt wird im Rahmen des Ausblicks dieser Arbeit noch weiter erörtert (siehe Abschnitt 7.3).

Über die Dienste der Beobachtung und Analyse, sowie auf parametrischer Basis auch die Generierung, können auch Entwicklungsmodelle in die Evolutionsunterstützung eingebettet werden. Dafür sind diese Entwicklungsmodelle in einer Wissenskomponente zu parsen und in eine dynamische EMF-Instanz zu laden. Dadurch können die Informationen dann über die entsprechenden Dienste verifiziert, analysiert oder angepasst werden. Beispielsweise wurden im Rahmen der Forschungstätigkeit dieser Arbeit für interdisziplinäre CPPS-Modelle nach Mund et al. [MJB+17] Fehlerraten zur Laufzeit bestimmt. Dazu wurden die Modelle über ein EMF-Meta-Modell eingelesen, der Beobachtungsdienst mit entsprechenden Aussagen ausgeführt und die EMF-Instanz entsprechend der Ereignisse dieser Aussagen verändert.

Aktive Steuerung der Koevolution

Durch die Möglichkeit, Wissen aus den Zuständen zu extrahieren, ist allerdings noch nicht zwingend sichergestellt, dass das notwendige Wissen für gegenwärtige und vor allem zukünftige Aufgaben auch vollständig dokumentiert wird. Denn dafür muss der Prozess der Koevolution aktiv gesteuert werden, um zielgerichtet Wissen aufzubauen und aktuell zu halten. Deshalb muss neben der Existenz der unterschiedlichen Dienste der Wissenskomponente dafür auch ein zusätzlicher Selbststeuerungmechanismus etabliert werden. Dieser bestimmt dann, wann die miteinander konkurrierenden Dienste verwendet werden. Dazu bedient sich die Systemarchitektur aus Konzepten der Welt von Softwareagenten. Für den zu steuernden Prozess liegt mit der Koevolution von Verhaltensmodelle ein klares Ziel vor. Deshalb wird zur aktiven Steuerung ein zielbasiertes Vorgehen gewählt, welches es ermöglicht, auch konkurrierende Ziele gegeneinander abzuwiegen. Diese wird im Rahmen der Systemarchitektur durch eine **Managementkomponente** implementiert (siehe Abbildung 4.6).

Unter den Rahmenbedingungen der hohen Dynamik von CPSen ist dabei ein robuster Ansatz zu favorisieren. Aus diesem Grund wird ein Konzept mit Priorisierung von Zielen gewählt, wobei die Prioritäten auf dem aktuellen Zustand des technischen Prozesses beruhen. Beispielsweise ist der Beobachtungsmodus genau solange aktiv bis eine Änderung im CPS festgestellt wird. Denn dann muss eine Entscheidung über einen Fehler oder eine Evolution getroffen werden. Abhängig von dieser Entscheidung wird die Beobachtung fortgesetzt oder eine Generierung vorgenommen.

Eine dafür passende Architektur ist ein zielbasierter Ansatz mit priorisierter Deliberation von Zielen nach dem Ansatz des *Belief-Desire-Intention (BDI)*. Bei diesem Ansatz gibt es Ziele, die durch Pläne erreicht werden können. Mit einer Zieldeliberation wird dabei – auch bei möglichen Zielkonflikten – jeweils ein Plan ausgewählt und in Abhängigkeit der vorherrschenden Bedingungen im CPS ausgeführt (*means-end reasoning*) [PBHL14]. Dabei sind diese Ziele als spezifisch annotierte Javaklassen im Managementservice implementiert.[7]

[7] Das dafür verwendete Konzept, was auf einem BDI-Ansatz aus reinem Java-Code basiert, wurde im Rahmen dieser Arbeit durch den Autor nicht vollständig selbst entwickelt, sondern die Entwicklung wurde konsultierend begleitet und dann der Ansatz auf die Problemstellung dieser Arbeit angewendet (siehe Pokahr, Braubach, Haubeck, Ladiges [PBHL14]).

Abbildung 4.8 erläutert das Grundprinzip der Zielhierarchie für diese gesteuerte Koevolution[8]. Das übergeordnete Ziel der Evolutionsunterstützung ist es dabei, eine vollständige Dokumentation in Evolutionsschritten sicherzustellen. Dieses übergeordnete wird im *BDI*-Ansatz als *erhaltende Ziel* (*maintain goal*) modelliert, was immer erfüllt sein muss. Dabei wird die Zielerfüllung durch an das Ziel annotierte Bedingungen gemessen. Diese geben an, wann das Ziel erreicht ist, oder wann es nicht mehr erfüllt ist. Beispielsweise ist es nicht mehr erfüllt, wenn eine Abweichung auftritt.

Die unterliegende Idee der weiteren Zielhierarchie ist es, dass die Dienste der Modelle zur Zielwiederherstellung verwendet werden können (*achieve goal*). Die Funktionalitäten, die ein Dienst dafür anbietet, werden als Pläne untergeordneter Ziele abgebildet. Die Ziele können nach ihren annotierten Vor- und Nachbedingungen zur Wiederherstellung einer aktuellen Dokumentation verwendet werden. Beispielsweise kann so eine Abweichung über eine Generierung gelernt werden, wodurch die Abweichung aufgelöst wird und eine aktuelle Dokumentation wieder erfüllt ist.

Wenn sich die spezifischen Bedingungen des Gesamtzustands in der durch die Wissenskomponente beobachteten Repräsentationskomponente ändern, wird eine Zieldeliberation durchgeführt. Diese hat zum Ziel, dass so der unter den Bedingungen erfolgversprechendste Dienst zur Ausführung ausgewählt.

Zu den Zielen der Koevolution von CPPSe gehört es auch, den Nutzer nach einer spezifischen Entscheidung zu fragen. Dabei wird der Nutzer als *Human in the Loop* verwendet. Dieser trifft beispielsweise die Entscheidung darüber, ob ein Fehler bzw. Evolutionsschritt jeweils *gewollt* oder *ungewollt* war.

Ziele und Pläne, wie die Generierung eines neuen Modells, können auch scheitern. Dies führt zu einer erneuten Zieldeliberation, welche dann den jeweils nächst niedriger priorisierten Dienst auswählt. Durch die Zielhierarchie wird somit auf eine robuste Art und Weise sichergestellt, dass immer der für die jeweilige Situation passendste Dienst ausgeführt wird.

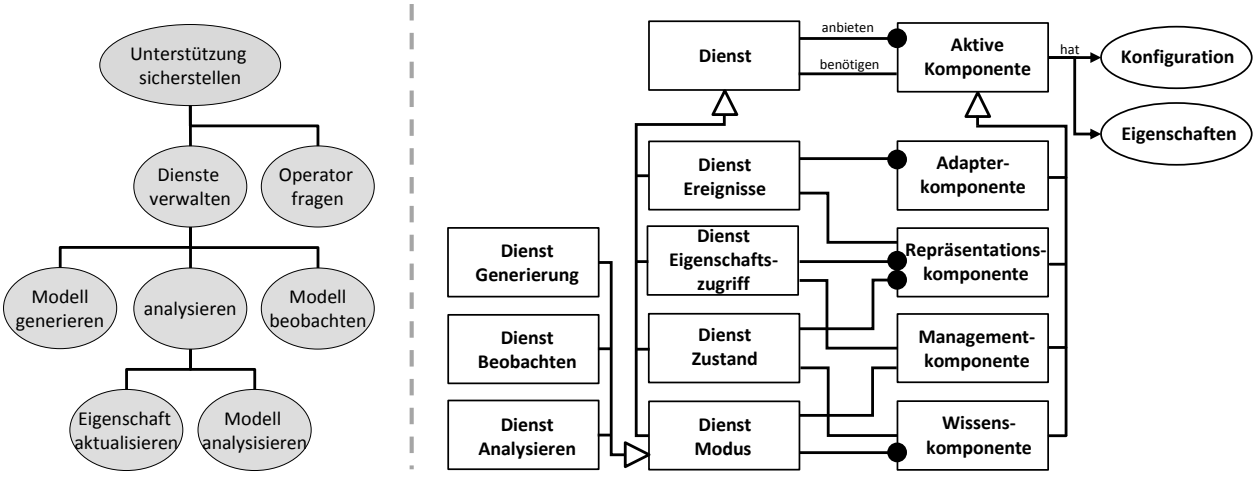


Abbildung 4.8: Links: Ziele für die Steuerung der Koevolution;
Rechts: Dienste und Komponenten für die Koevolution

[8]Vernachlässigung von Plänen und Beliefs

Zusammenfassung des Wissensmanagements

In diesem Abschnitt wurden Wissenskomponenten vorgestellt, die es ermöglichen beliebige Meta-Modelle in die Systemarchitektur einzubinden. Diese kapseln die Modelle gegenüber Zugriffen von außen indem eine eigene Ausführungsumgebung geschaffen wird, die nur durch Dienste von außerhalb der Komponente beeinflusst werden kann. Durch eine eigenständige aktive Steuerung und eine zielbasierte Ausführung durch Managementkomponenten wird eine vollständige Koevolution der Modelle zur Laufzeit gewährleistet. Dazu wurde ein Konzept für eine effizientere und effektivere Auswertung von Aussagen über die Zustände von CPSen vorgeschlagen und implementiert.

Zusammenfassend werden in der Systemarchitektur für die Koevolution, die auf der rechten Seite in Abbildung 4.8 aufgeführten Komponenten und Dienste verwendet. Eine Erweiterung der Systemarchitektur um weitere, über die Evolutionsunterstützung hinausgehende, Funktionalitäten bzw. Ziele ist in der Systemarchitektur explizit vorgesehen. Es bleibt festzustellen, dass durch die vorgestellte Systemarchitektur das objektive Bewusstsein, welches in der zweiten operativen Hypothese formuliert wurde, erreicht wird.

4.6 Cyber-physisches Evolutionsnetzwerk

Für ein vollständiges objektives Bewusstsein benötigt ein operatives System darüber hinaus auch noch Wissen über seine Umgebung und Verbindung zu anderen Systemen [Mah07]. Diese Tatsache wurde in der dritten Hypothese zu einem Netzwerk von wissenstragenden Komponenten formuliert, die auf den Austausch von Evolutionsschritten fokussiert.

Die in Abbildung 4.6 gezeigte wissenstragende Komponente enthält dafür weitere Komponenten auf Ebene der cyber-physischen Verbindung und insbesondere des Cyber-Netzwerks. Als erste solche Komponente muss die Methodik zur Bildung von Evolutionsschritten integriert werden. Die dafür vorgenommene Implementierung einer Evolutionsschrittkomponente zeigt Abschnitt 4.6.1.

Zusätzlich wird eine Art verteilter *Marktplatz* für Evolutionsschritte realisiert. Auf diesem Marktplatz ist es einer wissenstragenden Komponente möglich, Evolutionsschritte des eigenen CPS anzubieten und fremde Evolutionsschritte für das eigene CPS nutzbar zu machen (Abschnitt 4.6.2). Dafür kann auf einem Marktplatz sowohl passiv als auch aktiv nach unterschiedlichen Teilen von Evolutionsschritten gesucht werden. In der Systemarchitektur wird diese durch die Evolutionsunterstützungskomponente vorgenommen (Abschnitt 4.6.3).

4.6.1 Evolutionsschritte als zentrales Artefakt

Der Methodik eines schrittbasierten Evolutionsprozesses folgend sind Evolutionsschritte das zentrale Artefakt auf Ebene des Cyber-Netzwerks. Dabei sind zunächst Evolutionsschritte nach der vorgestellten Methodik zu erzeugen, was in der wissenstragenden Komponente in einer **Evolutionsschrittkomponente** implementiert ist. Diese *Aktive Komponente* implementiert folgende Funktionalitäten:

Reaktion: Wie im methodische Kapitel 3 gezeigt, basiert die Reaktion auf einer Modelldifferenz. Deshalb wurde das *Si-Lift Framework*[9] um die notwendigen Funktionalitäten und Konfigurationen für die Unterstützung von Evolutionsschritten und den darin verwendeten Modellen erweitert. Das *Si-Lift Framework* ist ein technisches Softwarewerkzeug

[9] http://pi.informatik.uni-siegen.de/Projekte/SiLift/

in Form eines *Eclipse*[10]*-Plug-ins*, welche den zugrundeliegenden Ansatz nach Kehrer et al [KKT11, KKOS12, Keh15] implementiert. Dabei kann *Si-Lift* sowohl manuell genutzt werden als auch programmatisch in Java eingebunden werden.[11] Das Framework und die zusätzlichen Funktionalitäten wurden im Rahmen dieser Arbeit mittels einer *Mircoagentenarchitektur* von *Aktiven Komponente* eingebunden bzw. implementiert.

Für die Anwendung auf die entwickelten Modellartefakte werden die in Abschnitt 3.4 entwickelten Meta-Modelle für CPPS- und CBCPS-Modelle über ihre *EMF-Ecore*-Beschreibung in die wissenstragende Komponente eingebunden. Das in Abschnitt 3.5.2 für Evolutionsschritte entwickelte ähnlichkeitsbasierter Verfahren zur Zuordnung ist als Erweiterung der Schnittstelle für so genannte *Modellelement-Matcher* von *Si-Lift* in Java implementiert worden. Das Verfahren geht dabei über die üblichen Matcher hinaus indem auch die Verwendung eines Modellkontextes statt eines gesamten Modells ermöglicht wird. Der Modellkontext wird dazu bei der Anwendung eines Evolutionsschrittes dem Matcher als Kontext übergeben. Das Liften von Modelldifferenzen verwendet die in Abschnitt 3.5.3 erläuterte Operationsmengen für CPPS- und CBCPS-Modelle. Die Operationen sind durch komplexe Editierregeln im *Henshin*-Framework[12] abgebildet. Bei *Henshin* handelt es sich um eine Transformationsprache für *EMF*-Modelle, die eine Suche nach Teilgraphen ermöglicht. Mit diesen Anpassungen kann die Reaktion eines Evolutionsschrittes über die *Post Processed Recognition Engine* von *Si-Lift* berechnet werden. Dazu werden die *EMF*-Modelle geladen, durch den Matcher zugeordnet und durch *Si-Lift* unter Verwendung der implementierten *Henshin*-Regeln für Operationen geliftet. Der Modellkontext wird durch den Matcher individuell ausgegeben und dem Editierskript beigefügt.

Ergebnis: Für die Berechnung des Ergebnisses wird der Analysedienst der Wissenskomponenten verwendet. Dieser wird über ein Dienstaufruf der Evolutionsschrittkomponente genutzt. Aus den Ergebnissen dieser Analyse können nach den in Abschnitt 3.5.4 gegeben Formeln die absoluten und relativen Differenzen berechnet werden.

Grund: Für die Bestimmung des Grunds ist die erkannte Abweichung von beobachteten und erwarten Ereignissen der Zustände zu finden. Die in Abschnitt 3.5.5 definierten Abweichungen basieren auf Fehlerfiltern nach Junker [Jun16]. In dessen Arbeit werden auch formale Ausdrücke dieser Fehlerfilter definiert, welche in dieser Arbeit in Ausdrücke auf den Zustandsdienst abgebildet werden. Dazu wird das Konzept von Aussagen verwendet und implementiert. Entsprechend werden bei jedem Fehler die Aussagen der definierten Gründe (Wegfall, Einfügen, Veränderung, Zeitaberration und Übergreifend) ausgewertet. Wenn die Aussagen erfüllt sind, liegt ein entsprechender Grund im Evolutionsschritt vor.

Evolutionsschritte: Nach der Bestimmung von Reaktion, Ergebnis und Grund wird der Evolutionsschritt mit dem gegebenen Besitzer erzeugt. Vorgänger, Nachfolger und etwaige Vererbungen werden bei ihrer Entstehung durch einen Callback-Mechanismus angehängt. Evolutionsschritte werden, wie alle zwischen den Komponenten ausgetauschten Ereignisse und Objekte der Architektur, als *JavaBeans* übertragen. Im Rahmen dieser Arbeit wurde an dieser Stelle die *Call-by-Reference*-Semantik bei Dienstaufrufen verwendet, sodass immer über Proxy-Objekte auf Evolutionsschritte der ursprünglichen wissenstragenden Komponente referenziert wird. Es wäre allerdings auch möglich, eine *Call-by-Value*-Semantik zu verwenden, wodurch allerdings Vor- und Nachfolger sowie Vererbungen auf dem Stand der Übertragung verbleiben würden.

[10] https://www.eclipse.org/

[11] Die Einbindung des *Si-Lift Frameworks* wurde im Rahmen der Forschungstätigkeiten zu dieser Arbeit in Kooperation mit der Universität Siegen, insbesondere Christopher Pietsch und Timo Kehrer, durchgeführt.

[12] https://www.eclipse.org/henshin/

4.6.2 Marktplätze aus wissenstragenden Komponenten

Um einen Selbstvergleich im Sinne der dritten Hypothese zu erreichen, erfordert es einer Verknüpfung von wissenstragenden Komponenten auf Ebene des Cyber-Netzwerks. Dafür sind wissenstragende Komponenten in die Lage zu versetzten, unbekannte Evolutionsschritte zu erhalten, zu vergleichen und zu evaluieren. Denn nur dadurch kann die im vorangegangenen Kapitel methodisch vorgeschlagene kooperative Evolution durch Vorschläge umgesetzt werden. In der vorliegenden Arbeit wird für diese Verknüpfung die bereits genannte Metapher eines Marktplatzes verwendet. Über diesen können Vorschläge vergleichbar zu einer Bibliothek von Änderungen, wie es beispielsweise für wiederverwendbare Module von Produktionssystemen vorgeschlagen wird [FFVH+12], verwaltet werden. Dies ermöglicht eine Erweiterung des Erfahrungshorizont von CPSen bezüglich einer möglichen Evolution [HCL+17].

Für die Realisierung eines derartigen Marktplatzes scheint auf den ersten Blick ein zentralisierter Ansatz sinnvoll, bei dem alle Evolutionsschritte in einem zentralen Repository gesammelt werden. Dort könnten sie bezüglich des Grunds, der Reaktion und des Ergebnisses analysiert werden. Allerdings stellt ein solches Repository einen *Single-Point of Failure* dar, benötigt zusätzliche Infrastrukturen und verursacht einen hohen administrativen Aufwand [HCL+17]. Des Weiteren birgt ein solcher Ansatz erhöhte Sicherheits- und Datenschutzrisiken. Diese treten bei geschäftsrelevanten Daten immer auf, könnten allerdings durch einen dezentralen Ansatz verringert werden, bei dem die Administration, Verteilung und Verantwortung dezentral verbleibt. Entsprechend wird ein dezentraler Ansatz gewählt.

Ein wesentlicher Wert der Evolutionsunterstützung wird durch die Struktur des Marktplatzes selbst geschaffen. Dies zeigt sich bereits jetzt schon bei Varianten- und Versionierungswerkzeuge während der Entwicklung. Denn auch dort werden zunehmend dezentrale Repositorys, wie z.B. *GIT*, verwendet, um die verteilte Entwicklung zu erleichtern. Bei steigender Anzahl von Teilnehmern ist, wie auch bei der Versionskontrolle und parallel arbeitenden Entwicklern, eine gute Skalierbarkeit der Systemarchitektur entscheidend.

Deshalb wird zum Austausch von Evolutionsschritten eine Peer-to-Peer-Architektur auf Ebene des Cyber-Netzwerks etabliert. Dabei können Peer-to-Peer-Netzwerke in unterschiedlichen Umgebungen eingesetzt werden und eine hohe Autonomie der Ausführung sowie das selbstbestimmte Bereitstellen von Informationen sicherstellen [CDK05]. Bezüglich nichtfunktionaler Eigenschaften bieten diese eine gute Skalierbarkeit und Widerstandsfähigkeit, insbesondere gegenüber dem Ausfall einzelner Teilnehmer [CDK05]. Diese ermöglicht einen Marktplatz mit einer unbeschränkten Anzahl von unabhängigen und gleichberechtigten wissenstragenden Komponenten.

Bezüglich der partizipierenden Teilnehmer und ihres Austausches von Evolutionsschritten sind zwei gegenläufige Effekte gegeneinander abzuwägen: Zum einen existiert zwischen zwei wissenstragenden Komponenten, die sehr ähnliche CPSe auf dem Marktplatz repräsentierten, eine hohe Korrelation bei der Anwendbarkeit von Evolutionsschritten. Denn Evolutionsschritte von ähnliche Verhaltensmodelle haben eine hohe Kompatibilität. Allerdings bieten zum anderen wissenstragende Komponente mit abweichenden Modellartefakten ungewöhnliche und dadurch auch innovativere Evolutionsschritte. Denn dort verwendete Lösungen wurden ggf. in einem abweichenden System noch nicht bedacht. Entsprechend unterliegt der durch die vorhandenen Evolutionsschritte aufgespannte Lösungsraum dem Spannungsfeld von vergleichbaren und passenderen Evolutionsschritten von ähnlichen CPSen und innovativeren und abweichenden Evolutionsschritten von verschiedenartigeren CPSen. Dieses Spannungsfeld gilt für alle Aspekte der Evolutionsschritte gleichermaßen.

Für einen Erfahrungsaustausch ist diese Feststellung allerdings positiv zu sehen, da damit das Potenzial der über den Marktplatz verteilten, ungenutzten Erfahrung erhöht wird. Mögliche Anwendungsszenarien umfassen dem Evolutionsverständnis der Arbeit folgend, Änderungen mit Erweiterung und Verbesserung, genauso wie die Wartung mit Erhaltung, Korrektur und Sanierung. Dies entspricht der im Wissensmanagement empfohlenen organisatorisch geleiteten Nutzbarmachung von Erfahrungen [Wil13]. Durch diese kann die Evolution vorhersehbarer und weniger riskant gemacht werden.

Austausch von Evolutionsschritten

Jede am Marktplatz partizipierende wissenstragende Komponente bietet ihre lokale Menge von Evolutionsschritten über einen Dienst der Evolutionsschrittkomponente an (vergleiche Abbildung 4.6). Der Methode der Evolutionsunterstützung folgend kann über den Marktplatz, der in Abbildung 4.9 gezeigte Austauschprozess realisiert werden.

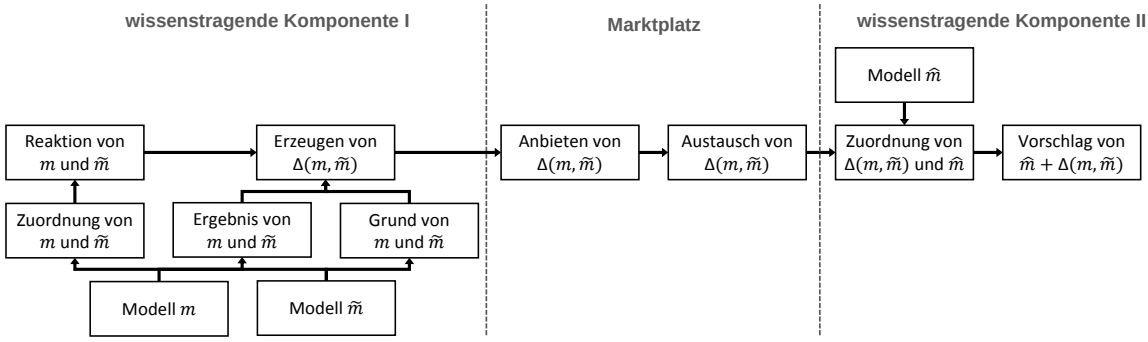


Abbildung 4.9: Austauschprozess für einen Marktplatz von Evolutionsschritten

Der gezeigte Austauschprozess geht von zwei wissenstragenden Komponenten mit den entsprechenden Funktionalitäten aus. Wenn die bereitstellende Komponente eine neue Evolution durch den Beobachtungsdienst der Wissenskomponente des Modells m erkennt, wird durch den Generierungsdienst ein neues Modellartefakt $\tilde{m}$ generiert. Aus diesem wird im Vergleich zum vorherigen Modell m durch die Evolutionsschrittkomponente ein neuer Evolutionsschritt $\Delta(m,\tilde{m})$ erstellt. Dieser hat die wissenstragende Komponente als Besitzer. Weiterhin enthält der Evolutionsschritt die klassifizierte Abweichung als Grund, die aus der Zuordnung $Z(m,\tilde{m})$ abgeleitet Reaktion sowie Eigenschaftsdifferenzen des Analysedienstes als Ergebnis.

Da beim Austausch mit Reaktion und Kontextinformationen sensible Parameter transferiert werden, verfolgt die Implementierung an dieser Stelle drei mögliche Schutzstrategien:

- Es wird nur die Wirkung der Reaktion übertragen. Dies beinhaltet als Modellkontext nur die Kontextinformationen der Modellelemente, die Eingangsparameter einer Operation der Wirkung sind. Dies exkludiert den für die Transformation der Reaktion nicht direkt verwendete Modellkontext, wie angrenzende Modellelemente, da diese nicht Teil der eigentlichen Wirkung sind. Dies ermöglicht die höchste Form des Schutzes des Anbieters eines Evolutionsschrittes. Allerdings erschwert das Vorgehen das Finden einer geeigneten Zuordnung auf Seiten des Nutzers dieses Evolutionsschrittes.
- Es werden neben der Wirkung auch die Kontextinformationen der Modellartefakte übertragen, die eine direkte Verbindung zu den Modellelemente haben, welche Eingangsparameter einer Operation der Wirkung sind. Entsprechend kann beim Nutzer des Evolutionsschrittes eine lokal begrenzte Ähnlichkeit bestimmt werden. Diese umfasst jedoch nicht die vollständigen Ereignisfolgen.

- Die letzte Strategie umfasst die Übertragung des vollständigen Modellkontextes der Modellelemente, die Eingangsparameter einer Operation der Wirkung sind. Dies sind alle vollständigen Ereignisfolgen, auf denen diese Modellelemente liegen. Dies bietet den geringsten Schutz des Anbieters, ermöglicht allerdings auch die bestmögliche Zuordnung. Im Rahmen der Evaluation der vorgeschlagenen Konzepte wurde daher in dieser Arbeit auch diese Strategie gewählt.

Evolutionsschritte und ihre Modellkontexte werden softwaretechnisch über einen Dienst der Evolutionsschrittkomponente auf dem Marktplatz angeboten. Die im Rahmen dieser Arbeit umgesetzte Realisierung dieses Dienstes verwendet eine *Blockchain*, die den verteilten Marktplatz realisiert. Dazu werden die einzelnen Evolutionsschritte wie Transaktionen behandelt und diese mittels der Blockchain unter allen Teilnehmern geteilt. Dieses Vorgehen wird im folgenden Abschnitt 4.6.2 weiterführend erläutert.

Ein so ausgetauschter Evolutionsschritt kann durch die Evolutionsschrittkomponente genutzt werden. Dafür wird eine Zuordnung zwischen dem Modellkontext des ausgetauschten Evolutionsschrittes und dem aktuellen Modell vorgenommen. Durch die Anwendung des Evolutionsschrittes kann dann ein Vorschlag abgeleitet werden und dem Nutzer dadurch eine Handlungsalternative unterbreitet werden. Die Handlungsalternative gibt an, wie das beobachtete CPS mit den Erfahrungen des anbietenden CPSs weiter evolviert werden kann. Sie umfasst dabei die Auswirkungen in einer modellbasierten Beschreibungsform und die prognostizierten Veränderungen der nichtfunktionalen Eigenschaften.

Softwaretechnisch werden die Vorschläge im Dienst der Evolutionsschrittkomponente bestimmt, der von der Evolutionsunterstützungskomponente genutzt wird. Dazu vergleicht zunächst der dort implementierte Matcher die Modelle einer Wissenskomponente mit dem Modellkontext des Editierskripts. Die so erzeugte Zuordnung dient zur Auflösung der Eingangsparameter des Editierskripts. Mit diesen aufgelösten Parametern kann das *SiLift*-Framework das Editierskript anwenden und ein durch den Evolutionsschritt evolviertes *EMF*-Modell erstellen. Durch die Nutzung des Dienstes der Analyse der zugehörigen Wissenskomponente werden dann die Eigenschaften der Handlungsalternative berechnet. Der im Rahmen dieser Arbeit realisierte Prototyp visualisiert dazu die Handlungsalternative über das *JGraphx*-Framework[13].

Der Austauschprozess zeigt, dass ein Marktplatz nur das Anbieten und den Austausch der Evolutionsschritte implementieren muss. Die Erzeugung sowie die Anwendung wird innerhalb der jeweiligen wissenstragenden Komponenten vorgenommen. Beides zusammen bestätigt den Aspekt einer systemunterstützten Evolutionsentscheidung, wie in der dritten Hypothese postuliert.

Blockchain-basierte Verwaltung von Evolutionsschritten

Mit einem verteilten Marktmarkt soll ein gesicherter und fairer Austausch zwischen gleichberechtigten Teilnehmern des Peer-to-Peer Netzwerkes ermöglicht werden. Die Arbeit nimmt dabei an, dass ein Evolutionsschritt als eine Transaktion von Wissen in CPSen zu verstehen ist. Dies geht mit der Sichtweise einher, dass durch einen Evolutionsschritt Erfahrungen transferiert werden. Eine Möglichkeit, solche Transaktionen zu verwalten, ist es dabei, dezentralen Kontobüchern oder Datenbanken von Transaktionen zu verwenden [MS15].

Softwaretechnisch werden diese Funktionalitäten durch das aufstrebende Gebiet der *Distributed-Ledger-Technologien* abgedeckt. Wie im Grundlagenkapitel dargelegt, haben auch Blockchains als kleinste Einheit jeweils Transaktionen, die beliebige Daten enthalten können

[13] https://github.com/jgraph/jgraphx

[PKBL18]. Deshalb stellt eine Blockchain eine Möglichkeit dar, einen verteilten Marktplatz unter den gegebenen Anforderungen der Evolutionsunterstützung zu realisieren indem Transaktionen zwischen verschiedenen Teilnehmern dezentral über eine Blockchain propagiert werden (vergleiche [MS15]). Dabei stellt die Blockchain jeweils einen Konsens über den Zustand des Austausches sicher. Dies entspricht im Sinne der Evolution dem Zustand des momentanen Evolutionsprozesses und damit aller Evolutionsschritte und ihrer erfolgten Vererbungen. Zusätzlich wird durch die kontrollierte Konsensbildung die Integrität der Evolutionsschritte erhalten.

Die Anforderungen könnten dabei auch durch andere verteilte Datenbanktechnologien erreicht werden. Allerdings ist gilt eine Blockchain als zukünftige Technologie für die Implementierung eines verteilten Marktplatzes [Sub18]. Diesbezüglich gibt es in den letzten Jahren verschiedene Ansätze und Beispiele Dienste oder Informationen über einen blockchain-basierten Marktplatz anzubieten (vergleiche [KET+17, SKCD16, BM16]). Als wesentliches Merkmal wird dadurch der direkte Austausch zwischen Systemen ohne zentrale Kontrolle abgewickelt, was zu einem Erhalt von Dateneigentum, Transparenz und Vertrauen einhergeht [MS15, HBL+18].

Für die hier vorgeschlagene Evolutionsunterstützung wird auf vorhandene Technologien zurückgegriffen, die Blockchains für beliebige Transaktionen realisieren. Zur Implementierung verwendet die Arbeit die *Cadeia* Blockchain[14] auf Basis von *Aktiven Komponenten*.[15] Entsprechend wird jede wissenstragende Komponente mit einer Blockchain-Komponente erweitert (vergleiche Abbildung 4.6).

Die daraus resultierenden Funktionalitäten und ihre Interaktionen im Rahmen der vorliegenden Arbeit zeigt Abbildung 4.10. Als erster Schritt erzeugt die wissenstragende Komponente Evolutionsschritte, die dann über deren Dienst dem *Evolutionsschrittpool* der Blockchain-Komponente hinzugefügt werden. Diese ist in dieser Arbeit durch ein *Merkle tree* implementiert (siehe [Mer87]). Dabei werden durch Evolutionsschrittpool nur vollständige Evolutionsschritte anerkannt. Wenn ein neuer Evolutionsschritt dem Pool hinzugefügt wird, wird dieser an alle verfügbaren Blockchain-Komponenten und damit wissenstragenden Komponenten übertragen. Ein solches Verfahren sichert idealerweise, dass jede wissenstragende Komponente die gleichen Evolutionsschritte in ihrem Pool hat.

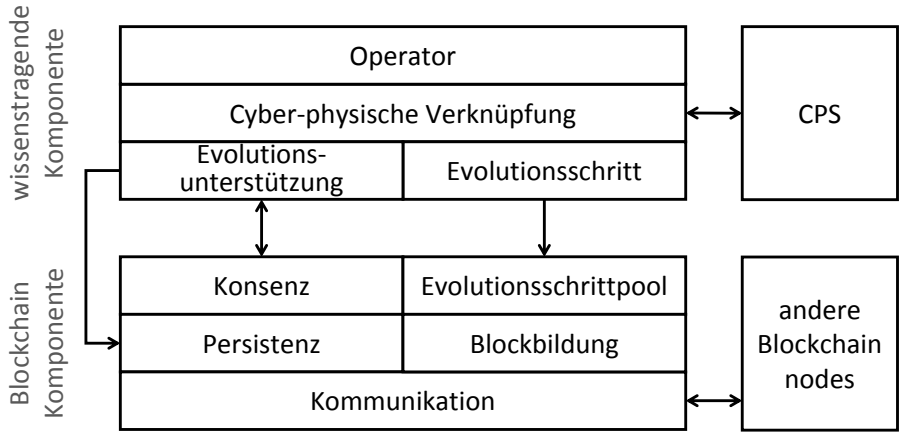


Abbildung 4.10: Funktionalitäten der Blockchain-Komponente

[14] https://www.cadeia.org/

[15] Die *Cadeia* Blockchain wurde im Rahmen eines Studentenprojektes während der Forschungsarbeiten zu dieser Arbeit entwickelt und implementiert.

Aus dem Evolutionsschrittpool werden neue Blöcke gebildet, welche in der Blockchain durch die teilnehmenden Komponenten vorgeschlagen werden (*Blockbildung*). Blöcke werden über kryptographische Hashes miteinander verknüpft, wodurch die Struktur der Blockchain unveränderlich festgelegt wird [PKBL18]. Um langfristig denselben, validen Zustand der Blockchain und damit des Evolutionsprozesses in jeder wissenstragenden Komponente zu erhalten, wird ein Konsens-Algorithmus verwendet. Über diesen entscheiden die Blockchain-Komponenten kooperativ, welche Blöcke jeweils der Blockchain anhängt werden. Im Rahmen der Implementierung dieser Arbeit wurde dafür der *Tendermint Proof-of-Stake-Algorithmus* verwendet (siehe [Buc16]). Dieser Algorithmus nutzt eine Menge von Validatoren, die sich rundenbasiert über den Zustand abstimmen. Dabei werden in dieser Arbeit alle beim Start des Netzwerks vorhandenen wissenstragenden Komponenten als Validatoren genutzt. Dadurch, dass in jeder Runde neue Blöcke vorgeschlagen und der Blockchain angehängt werden, entsteht, entgegen von kettenbasierten Algorithmen, nie eine Verzweigung der Blockchain [PKBL18].

Wenn ein Block durch die Validatoren abgestimmt hinzugefügt wird, wird dieser in der Blockchain-Komponente gespeichert (*Persistenz*). In der im Rahmen dieser Arbeit vorgenommenen Implementierung wird dazu die eingebettete Datenbank der *Cadeia* Blockchain verwendet.[16] Als Resultat besitzen alle Blockchain-Komponenten die selbe abgestimmte Blockchain. Dies entspricht im Sinne dieser Arbeit allen Evolutionsschritten und damit einer vollständig abgestimmten Kopie des Evolutionsprozesses.

4.6.3 Passive und Aktive Unterstützung des Nutzers

Der durch die Blockchain verteilt vorliegende Evolutionsprozess kann genutzt werden, um dem Nutzer Vorschläge bezüglich einer Fortentwicklung oder Wartung seines CPSs vorzulegen. Dies wird der Systemarchitektur folgend durch die **Evolutionsunterstützungskomponente** realisiert (vergleiche Abbildung 4.6). Diese setzt damit die eigentliche Unterstützung des Nutzers um. Beim Zusammenspiel zwischen wissenstragender Komponente und dem Nutzer sind dabei prinzipiell zwei Arten von Unterstützung für die Evolution von CPSen denkbar:

- Eine **passive Unterstützung**, bei dem der Nutzer direkt über die wissenstragende Komponente einen seinen Präferenzen entsprechenden Evolutionsschritt anfragt und bereitgestellt bekommt.
- Eine **aktive Unterstützung**, bei der die wissenstragende Komponente die Menge von Evolutionsschritten aktiv nach passenden Evolutionsschritten durchsucht und diese dem Nutzer von sich aus anbietet.

Nachfolgend wird der Nutzen der Evolutionsunterstützung anhand dieser beiden Typen für die drei wesentlichen Aspekte von Evolutionsschritten beschrieben.

Passive Unterstützung

Die Evolutionsunterstützungskomponente realisiert eine passive Unterstützung indem der Nutzer eine Bewertung für die vorhandenen Evolutionsschritte anstößt. Die Komponente bestimmt daraufhin die Bewertung für jeden möglichen Vorschlag (siehe Abschnitt 3.6.1). Daraufhin kann einer oder mehrere Vorschläge auswählt werden. Dabei wird in der für die Evaluation implementierten Lösung der Vorschlag von allen in der Blockchain gespeichertem Evolutionsschritte bestimmt. Für größere Netzwerke wäre an dieser Stelle ein effizienteres Vorgehen bei den zu evaluierenden Evolutionsschritte notwendig. Die Arbeit diskutiert ein solches Vorgehen im Ausblick in Abschnitt 7.3.

[16] *LevelDB Datenbank*: http://leveldb.org/

Die Bewertung von Vorschlägen hängt bei einer passiven Unterstützung hauptsächlich von den Szenarien ab, in der ein Nutzer die Unterstützung anfordert. Unter dem Evolutionsverständnis dieser Arbeit bestehen diese Szenarien sowohl in Änderungen als auch in der Wartung. Bei der modellbasierten Unterstützung ist dabei entscheidend, ob dem Nutzer ein Modell der Änderungen bereits vorliegt oder er die Evolution ohne ein solches durchführt. Dabei gibt es zum einen eine geplante Evolution, welche einen vollständigen Entwicklungsprozess durchläuft. Dies umfasst eine manuelle, vorherige Anpassung des momentanen Verhaltensmodell. Zum anderen gibt es eine Ad-hoc Evolution, die oft unter erheblichem Druck durchgeführt wird und bei der keine vorherige Anpassung des Modells in einem Entwicklungsprozess stattfindet (vergleiche [VHFST15, LWA+13]).

Tabelle 4.1 zeigt die, durch die Evolutionsunterstützungskomponente realisierte Unterstützung für diese beiden Szenarien. Diese wird nachfolgend anhand der drei wesentlichen Aspekte eines Evolutionsschrittes (Reaktion, Grund, Ergebnis) näher erläutert.

Tabelle 4.1: Nutzen der passiven Unterstützung

Aspekt	**geplante Änderungen**	**Ad-hoc Änderung**
Reaktion	vergleichen	vorschlagen
Grund	vergleichen	auswählen
Ergebnis	vergleichen	auswählen, abschätzen

Geplante Evolution:

Reaktion: Durch den gegebenen Vorschlag kann der Nutzer bewerten, ob die Reaktion des vorgeschlagenen Evolutionsschrittes der geplanten Änderung im Modell entspricht. Dadurch wird es für den Nutzer möglich, einen Vergleich seiner geplanten Änderung und der vorgeschlagenen Änderung auf Basis des manuellen Modells und des vorgeschlagenen Modells vorzunehmen. Beispielsweise könnte dadurch festgestellt werden, ob vergleichbare Verhaltensänderungen in anderen CPSen bereits vorgenommen wurden.

Grund: Anhand des Grunds kann der Nutzer bestimmen, ob es Vorschläge gibt, die aufgrund der selben Abweichungen vorgenommen wurden. Dies ermöglicht abzuschätzen, ob der Vorschlag ein ähnliches Problem eines CPSs betrachtet. Dadurch könnten z.B. CPSe identifiziert werden, bei denen ähnliche Abweichungen auftraten. Denn dies lässt auf vergleichbare und damit nutzbare Evolutionsschritte schließen.

Ergebnis: Das erwartete Ergebnis der manuell vorgenommenen Änderung des Modells und das Ergebnis der vorgeschlagenen Änderungen können miteinander verglichen werden. Dadurch kann geprüft werden, ob durch andere CPSe Änderungen vorgenommen wurden, die ein besseres Ergebnis versprechen.

Ad-hoc Evolution:

Reaktion: Der gegebene Vorschlag kann als Blaupause dafür dienen, wie eine Änderung des CPSs durchgeführt werden sollte. Der Nutzer kann verstehen, wie das Verhalten nach dem angestrebten Evolutionsschritt aussieht und welches Verhalten und welches Ergebnis daraus resultiert. Des weiteren können über den Modellkontext CPSe ausgemacht werden, die sich in einer vergleichbaren Situation befunden haben. Dies ermöglicht weitere Evolutionsschritte zu finden, die in einer vergleichbaren Situation eine Verbesserung erzielt haben. Beispielsweise kann für CPPSe das nach der Evolution vorliegende Routing von Werkstücken

analysiert werden. Dies ermöglicht eine fundiertere Entscheidung bei der Umsetzbarkeit von Änderungen.

Grund: Eine weitere Unterstützung bei der Einschätzung der Umsetzbarkeit ist der Grund, der zu einem Evolutionsschritt geführt hat. Dieser gibt Auskunft, ob eine Abweichung, die zu einer Ad-hoc Änderung geführt hat, einen der momentanen Situation ähnlichen Ursprung hat. So könnten Evolutionsschritte mit vergleichbaren Zeitaberration auf vergleichbare Gründe und damit auf eine ähnliche Fortführung der Evolution hinweisen.

Ergebnis: Über das Ergebnis des Evolutionsschrittes können die verschiedenen Handlungsalternativen abgewogen werden indem die Auswirkung dieser Evolution ersichtlich wird. Dies erfolgt indem der Nutzer beispielsweise erfährt, welche Auswirkungen ein vergleichbarer Evolutionsschritt auf die Flexibilität und Performanz des CPSs hatte. Dadurch kann das Ergebnis, ebenso wie auch die Reaktion, dafür genutzt werden, unbekannte und nicht bedachte Auswirkungen einer Evolution besser abzuschätzen.

Aktive Unterstützung

Die wissenstragende Komponente kann auch ohne den Aufruf durch den Nutzer aktiv agieren. Diese Art von Unterstützung besitzt ein großes Potenzial, das in dieser Arbeit nur in Teilen ausgeschöpft wird, da nur eine beschränkte Anzahl von Möglichkeiten untersucht wurde.

Tabelle 4.2: Nutzen der aktiven Unterstützung

Aspekt	**lokal**	**Netzwerk**
Reaktion	reaktiv	proaktiv
Grund	reaktiv, vorab	proaktiv
Ergebnis	proaktiv	proaktiv

Für die Arbeit wurden auf Basis des Mikroagentenarchitektur von *Aktiven Komponenten* zwei dieser Funktionalitäten prototypisch implementiert und näher untersucht: Dies ist einerseits eine Detektierung von lokalen Situationen der wissenstragenden Komponente. Dazu werden die Dienste der Systemarchitektur selbst beobachtet indem die Instrumentalisierung der Dienstbeobachtung, die für CBCPSe verwendet wurde, auf die wissenstragende Komponente selbst angewendet wird. Dies ermöglicht es unter anderem, den Wechsel in der Dienststeuerung der Koevolution wahrzunehmen. Wenn Wechsel auftreten können dadurch proaktiv Vorschläge zur Verbesserung der vorliegenden Situation gesucht werden.

Neben der lokalen Situation werden zudem die Mengenverhältnisse der vorliegenden Evolutionsschritte im Netzwerk betrachtet. Dazu wurde für den Grund, die Reaktion und das Ergebnis jeweils eine Regel definiert, welche die Ausprägung dieses Aspektes bei allen Evolutionsschritten des Netzwerks untersucht. In der Evolutionsunterstützungskomponente wird dann mithilfe dieser Regeln jeder neue Evolutionsschritt untersucht. Dabei wird stetig das in Formel 4.1 angegebene Verhältnis pn_{Aspekt} berechnet. Dieses Verhältnis beschreibt die Menge von Evolutionsschritten, die die Regel erfüllen (p_{Aspekt}), zur Gesamtzahl der Evolutionsschritte (n). Dabei müssen mindestens n_{min} Evolutionsschritte vorliegen.

$$pn_{Aspekt} = \begin{cases} \textbf{if } n < n_{min} \textbf{then } 0 \\ \textbf{else then } \dfrac{p_{Aspekt}}{n} \end{cases} \tag{4.1}$$

Dabei löst die Evolutionsunterstützungskomponente über das Verhältnis genau dann die Erstellung eines Vorschlags für alle diese Regeln erfüllenden Evolutionsschritte aus, wenn das Verhältnis pn_{Aspekt} zum ersten Mal (wieder) über einem dafür definierten Wert θ_{Aspekt} liegt. Der Nutzen dieser zwei Möglichkeiten einer aktiven Unterstützung wird abstrakt in Tabelle 4.2 zusammengefasst und wird anschließend näher ausgeführt.

Lokale Situation:

Reaktion: Der Vorschlag, basierend auf einer lokalen Situation, ermöglicht es, Handlungsalternativen bezüglich der aufgetretenen Situation aufzuzeigen. Dadurch reagiert die wissenstragende Komponente reaktiv auf Ereignisse, die sich durch eine Beobachtung des CPSs in den Verhaltensmodellen des CPSs niederschlägt. Beispielsweise wird so beobachtet, ob ein neues Modellartefakt generiert wurde. Dadurch ist es möglich, bei einer Generierung eines neuen Modells direkt potentiell folgende Evolutionsschritte aufzuzeigen indem dem Nutzen für das neue Modell passende Vorschläge unterbreitet werden.

Grund: Die Identifikation des Grunds eines Evolutionsschrittes kann, entgegen der Reaktion und des Ergebnisses, auch schon vor der Erkennung einer Evolution nützlich sein. Denn der Grund kann Aufschluss darüber geben, was für eine Situation im CPS vorherrscht. Beispielsweise können dann für eine bestimmte Abweichung proaktiv Vorschläge zur Behebung solcher Abweichungen angezeigt werden.

Ergebnis: Das Ergebnis kann reaktiv darüber Aufschluss geben, wie sich Situationen auf die Eigenschaften des CPSs auswirken. Durch den Vorschlag von Evolutionsschritten können bei einer Veränderung direkt Gegenmaßnahmen vorgeschlagen werden, die dabei auch mit den Modellartefakten quantifiziert werden können. Beispielsweise ist es so möglich, bei Performanzeinbußen eines Engpasses nach Evolutionsschritten zu suchen, die die technische Komponente des Engpasses beschleunigen können.

Evolutionsschritte im Netzwerk:

Reaktion: Für die Reaktion wurde im Rahmen dieser Arbeit eine Regel prototypisch implementiert, die jeweils bestimmt, ob eine Erweiterung des Transportsystem von CPPSen vorgenommen wurde. Dafür wird immer dann eine Auswertung ausgelöst, wenn die vorangegangenen Evolutionsschritte Operationen enthielten, welche ein Hinzufügen von Ereignissen an Transportbändern vorsehen. Dadurch kann beispielsweise bestimmt werden, dass im Netzwerk bei vielen Komponenten eines CPPSs zusätzliche Transportbänder eingesetzt wurden. Daraufhin kann für das beobachtende CPS ein vergleichbarer Evolutionsschritt vorgeschlagen werden.

Grund: Bezüglich des Grunds wurde im prototypisch implementierten Netzwerk u.a. untersucht, ob eine Zeitaberration bei spezifischen technischen Komponenten vorliegt. Dadurch kann beispielsweise bestimmt werden, dass sich die Antwortzeit einer CBCPS-Komponente im Netzwerk reduziert hat und so ggf. ein Evolutionsschritt angezeigt werden, der aufgrund dieser Reduktion vorgenommen wurde.

Ergebnis: Bezüglich des Ergebnisses wurde die Produktionsdauer von CPPSen in allen eingehenden Evolutionsschritten untersucht. Wenn viele Evolutionsschritte eine erhöhte Produktionsdauer aufweisen, werden die dafür ursächlichen Evolutionsschritte angezeigt.

Zusammenfassung des Evolutionsnetzwerks

In diesem Abschnitt wurde eine kooperative Evolutionsunterstützung durch ein verteiltes Netzwerk vorgestellt. Dazu wurde eine Systemarchitektur vorgeschlagen, die es erlaubt Evolutionsschritte auf vergleichbare Systeme anzuwenden. Diese ermöglicht, Evolutionsschritte zwischen auch entfernten Komponenten im Cyber-Netzwerk auszutauschen, um die eigene, lokale Evolution für den Nutzer zu erleichtern und zu verbessern. Dazu wurde ein Austauschprozess durch einen blockchain-basierten Marktplatz realisiert, der den Evolutionsprozess dezentral abbildet. Dadurch konnte ein auf Nutzeranfragen reagierendes als auch ein durch Informationsaustausch proaktiv agierendes Unterstützungssystem implementiert werden.

An dieser Stelle sei darauf hingewiesen, dass diese Evolutionsunterstützung den Einschränkungen der im vorherigen Kapitel erläuterten Methodik unterliegt. Dennoch kann durch die entwickelte Systemarchitektur die dritte operative Hypothese einer kooperativen Unterstützung bestätigt werden.

4.7 Zusammenfassung

Zusammengefasst wurden im operativen Kapitel drei Hypothesen formuliert, um eine Systemarchitektur für CPSe zu entwickeln, die die Evolutionsmethodik auf den verschiedenen Ebenen eines CPSs softwaretechnisch umsetzt.

Dazu wurden zunächst, zur Bestätigung der ersten Hypothese, die verschiedenen Ebenen eines CPSs bezüglich der Evolutionsmethodik erörtert und es wurde über Kategorien des CRI-Modells eine Beschreibung für spezifische CPSe aus den entsprechenden Domänenmodellen abgeleitet (Abschnitt 4.2). Diese dient dazu, die Evolutionsunterstützung allgemein den Ebenen von CPSen zuzuordnen und die dadurch realisierten Informationsströme darzustellen. Mit der Spezifizierung der Kategorien des CRI-Modells hat der Nutzer die Möglichkeit, Evolutionsschritte nicht nur in allgemeinen Aspekten, sondern auch systemspezifisch zu beschreiben.

Zur Bestätigung der zweiten Hypothese wurde in Abschnitt 4.3 die Systemarchitektur einer wissenstragenden Komponente eingeführt. Diese verwendet Adapterkomponenten um unterschiedliche heterogene Quellen von CPSen anzubinden und in einer Hierarchie von Repräsentationskomponenten konsistent zu verwalten (Abschnitt 4.4). Mit Hilfe von Wissenskomponenten wurden Laufzeitmodelle integriert, die eine Systemarchitektur für die Koevolution von Artefakten bereitstellt (Abschnitt 4.5). Dazu wurde ein zielorientierter Prozess vorgeschlagen, der es einer wissenstragenden Komponente erlaubt, die Generierung, Beobachtung und Analyse über Dienste autonom zu steuern und zu überwachen. Dies ermöglicht Modelle als Laufzeitartefakte konsistent im Bezug auf verschiedenen Ereignisquellen (halb)automatisiert zu koevolvieren. Die Systemarchitektur ist dabei in anderen Kontexten wiederverwendbar, sodass die gekapselten Laufzeitmodelle auch für andere Ziele als die Evolutionsunterstützung, wie z.B. die Selbstoptimierung oder die Selbstheilung, eingesetzt werden können.

Eine kooperative Evolutionsunterstützung zur Erfüllung der dritten Hypothese wurde schließlich auf Ebene des Cyber-Netzwerks vorgeschlagen und prototypisch realisiert (Abschnitt 4.6). Dazu wurden Evolutionsschritte selbstständig über einen verteilten, blockchain-basierten Marktplatz ausgetauscht. Dadurch kann eine dem CPS zugehörige wissenstragende Komponente jeweils Evolutionsschritte sowohl passiv als auch aktiv vorschlagen, um so den Nutzer bei seiner operativen Arbeit durch ähnliche Erfahrungen aus dem Netzwerk zu unterstützen. Diese Unterstützung gilt für alle Aspekte einer Evolution und bietet vielfältige Möglichkeiten Änderungen und ihre einzelnen Aspekte zu untersuchen.

5 Verwandte Forschung zum Thema Evolutionsunterstützung

Für die unterschiedlichen Forschungsgegenstände der Arbeit existieren eine Reihe an Lösungsansätzen, die zum Vorgehen in dieser Arbeit vergleichbar sind oder auch in dieser Arbeit genutzt bzw. erweitert werden. Zur Vorstellung dieser verwandten Forschung werden zunächst in Abschnitt 5.1 Lösungsansätze zu Taxonomien von Änderungen und Abweichungen erläutert. Anschließend werden in Abschnitt 5.2 Ansätze zu der Koevolution von Modellen erläutert und in Abschnitt 5.3 vergleichbare Ansätze der Evolutionsunterstützung über Netzwerke vorgestellt. Die Arbeit stellt in jedem Abschnitt jeweils diesem naheliegende Forschungsgebiete vor, erläutert ausgewählte vergleichbare Lösungsansätze und setzt diese mit den in dieser Arbeit vorgeschlagenen Konzepten in Beziehung.

5.1 Taxonomien von Evolution

Erstes Untersuchungsgegenstand dieser Arbeit ist eine Evolutionsbeschreibung.

5.1.1 Taxonomien von Änderungen

Die in dieser Arbeit vorgenommene Beschreibung formuliert eine *Taxonomie* für Änderungen. Eine der bekanntesten und in den Grundlagen bereits eingeführte Taxonomie geben Buckley et al. [BMZ+05]. Diese beschreibt Änderungen über die vier Aspekte *wann*, *wo*, *was* und *wie*. Darauf aufbauend nehmen Chapin et al. [CHK+01] eine mehr betriebswirtschaftliche Sichtweise ein indem sie auch die Aspekte des *warum* und dem *wer* einer Änderung aufnehmen. Diese allgemeinen Aspekte nimmt die Arbeit auf und spezifiziert Ausprägungen für Evolutionsschritte und Kategorien für CPSe. Bezüglich dieser Spezifizierung unterschiedet sich diese Arbeit von der vorgestellten allgemeinen Taxonomie, da hier nicht auf eine allgemeingültige Beschreibung beliebiger Änderungen, sondern auf eine Operationalisierung für die Evolutionsunterstützung auf Basis von Modellen fokussiert wird. Dennoch ist die dadurch erreichte Beschreibung nicht unabhängig von den für allgemeine Taxonomien definierten Ausprägungen zu sehen. Denn die vorgeschlagene Beschreibung operationalisiert und vertieft diese allgemeinen Ausprägungen. So wird z.B. die Auswirkung durch Buckley et al. [BMZ+05] nur in klein und schwer unterteilt und für die Beschreibung von Evolutionsschritten durch Eigenschaftsdifferenzen quantifiziert.

Neben diesen allgemeinen Taxonomien von Änderungen gibt es weitere Klassifizierungen, die jeweils unterschiedliche Schwerpunkte besitzen. In der Regel wird dabei entweder auf Änderungen des Quellcodes oder solche an der Architektur fokussiert. Diese spezifischeren Taxonomien werden durch William et al. [WC10] zusammenfassend vorgestellt. Die Taxonomien unterscheiden sich in den untersuchten Aspekten im Sinne von Chapin et al. [CHK+01] und der betrachten Architekturelemente, Eigenschaften und Auswirkungen (vergleiche [WC10]). Beispielsweise beschäftigen sich Fluri et al. [FG06] mit Änderungen, die aus Differenzen von Quellcode-Repositorys abgeleitet werden können. Und Lehnart et al. [LR+12] betrachten im Rahmen einer Auswirkungsanalyse autonome und kombinierte Änderungen des Quellcodes anhand von Graphoperationen. Generell fokussieren diese Taxonomien auf den *was*-Aspekt von Änderungen, wobei dies in der Regel anhand des Quellcodes, aber in Teilen auch von Anforderungen oder Softwaretests untersucht wird. Die Beschreibung von Evolutionsschritten versucht

hingegen die allgemeinen Aspekte über eine gemeinsame Grundlage von Ereignismodellen und Modelldifferenzen zu beschreiben. Dabei sollen die Aspekte insbesondere über Ereignisse maschinenlesbar erfassbar sein.

Auch für das Vorgehen bei der Evolution gibt es spezifische Taxonomien. Diese fokussieren damit auf den *wie*-Aspekt von Änderungen. Insbesondere für CPPSe zeigen Ladiges et al. [LHFL14a] und Vogel-Heuser et al. [VHDF+14], dass Änderungen in der Evolution auch entlang ihrer Artefakte definiert werden können. Dazu untersuchen die Autoren, in welcher Reihenfolge die Artefakte bei einer Änderung angepasst werden. Dieser Ansatz wurde im Rahmen der Forschungstätigkeit initial mit definiert und dient als theoretische Grundlage, für die in dieser Arbeit zur Unterstützung entwickelte Methode. Die Unterstützung wird dabei allerdings nicht auf Basis von Modellen explizit beschrieben, sondern die Erstellung entsprechender Modelle ist das Ziel der Verwendung dieser Beschreibung.

5.1.2 Taxonomien von Abweichungen

Neben Änderungen gibt es in der Literatur Taxonomien von Abweichungen des Verhaltens (*anomaly*). Zur vorliegenden Arbeit vergleichbar beschreiben Mazel et al. [MFF14] Abweichungen für den Netzwerkverkehr in verteilten Systemen. Dabei spezifizieren die Autoren Abweichungen über Klassen und Unterklassen, was den in dieser Arbeit verwendeten Dimensionen und Kategorien der *was*, *warum* und *wo* Aspekte einer Änderung entspricht. Wie in anderen Kategorisierungen von Chandola et al. [CBK09] benennen die Autoren dabei Klassen für Abweichungen, wie punktbasierte oder kontextabhängige Abweichungen. Allerdings werden diese Taxonomien nur kategorisiert und nicht für die Unterstützung operationalisiert. Weiterhin bleiben die Klassen an den Betrachtungsgegenstand gebunden. Dies ist für die generischen Dimensionen der in dieser Arbeit vorgestellten Beschreibung nicht gegeben, da diese sich an den allgemeinen Aspekten von Änderungen orientiert und dann über Kategorien eines Domänenmodells individuell spezifiziert werden.

Vergleichbare allgemeingültige Dimensionen finden sich in Taxonomien von Mirkovis et al. [MR04] und Tobergte et al. [TC13]. Mirkovis et al. beschreiben dabei die Intention und den Trigger im Rahmen von *Denial-of-Service*-Angriffen. Tobergte et al. untersuchen den Grund, den Ort und die Auswirkung von fehlerhaften Dienstaufrufen über fest definierte Klassen. Diese Dimensionen sind ähnlich zu der im Rahmen dieser Arbeit entwickelten Beschreibung, wobei diese letztlich auf die operationalisierte Beschreibung während der Evolution fokussiert und nicht auf die Klassifizierung von Abweichungen. Die für diese Arbeit interessanteste Taxonomie nehmen Avizienis et al. [ALR04] vor, da die Autoren Abweichungen über eine Kausalkette definieren. Dabei verursacht ein Defekt (*fault*) einen Fehler (*error*), der dann zu einem Ausfall (*failure*) führt. Dabei stützen die Autoren ihre Taxonomie, wie diese Arbeit auch, auf beobachtbare Ereignisse. Allerdings versuchen sie nicht, das fehlerhafte Verhalten für eine Unterstützung zu nutzen.

Die Erkennung von Evolution hat große Überschneidungen zum Forschungsbereich der Fehlererkennung und -diagnose, die insbesondere bei industriell geprägten CPSen eingesetzt werden. Eine Übersicht über diese Ansätze geben Isermann [Ise06] und Niggermann et al. [NL15] bezüglich CPPSe. Diese Ansätze können als Teil ihrer Erkennungsmethode eine operationalisierte Beschreibung dieser Abweichungen enthalten. Allerdings basieren diese Ansätze in der Regel auf spezifischen Domänen und fokussieren nicht auf eine domänenunabhängige Beschreibung von Änderungen. Dies hat den Grund, dass dadurch spezifische Phänomene der Anwendungsdomäne erkannt werden sollen und so keine allgemeine Methode für Änderungen von Interesse ist. Beispielsweise definieren Maier et al. [MNE15], ebenso wie die vorliegende

Arbeit, Fehler als Differenz zwischen einem operativ beobachteten Verhalten und einer Modellspezifikation. Und Roth et al. [RLL11] beschreiben Abweichungen über Residuen, um die Art oder den Ort von Fehlern zu identifizieren. Die Beschreibungen werden allerdings auf einem sehr spezifischen und niedrigen Abstraktionsniveau vorgenommen. Weiterführenden, allgemeine Aspekten der unterliegenden Änderung, wie z.B. der Grund oder die Auswirkungen, werden in diesen Ansätzen nicht betrachtet.

Eine methodische Ähnlichkeit der Beschreibung einer spezifischen Domäne hat die Evolutionsbeschreibung von Ciraci et al. [CvA07]. Diese nehmen auch eine Spezifizierung anhand von abstrakten Domänenmodellen vor, was in dieser Arbeit vergleichbar für CBCPSe und CPPSe durchgeführt wird. Die Evolutionsbeschreibung diente als Inspiration der spezifischen Beschreibung im Kontext der Evolution, wobei Ciraci et al. keine Beschreibung von Änderungen vornehmen.

5.2 Methoden zur Koevolution in CPSen

Als zweiter Untersuchungsgegenstand wird die Koevolution von Zuständen des operativen Systems und einem Modell betrachtet.

5.2.1 Modellbasierte Entwicklung

In der modellbasierten Entwicklung werden Modelle verwendet, um das Verhalten oder die Anforderungen eines Systems für die Entwicklung (formalisiert) abzubilden. Diese können auch für die Evolution eingesetzt werden, um u.a. Eigenschaften eines Systems zu verifizieren. Als Beispiel zur Evolutionsunterstützung von CPPSen stellen Legat et al. [LMC+14] ein interdisziplinäres Modell mit einer formalen Beschreibung der Anforderungen vor. Bei dieser Art von Ansätzen wird in der Regel das Modell vor der Implementierung erstellt, dann eine Verifikation mit beispielsweise formalen Ansätzen der Modellprüfung (*model checking*) durchgeführt und bei erfolgreicher Verifikation wird dann das Modell implementiert. Dabei können diese Modelle auch Eigenschafte, wie die der Performanz, der Wartbarkeit und der Flexibilität, abbilden.

Modellgenerierende Ansätze erweitern diese Methoden durch eine weitestgehend automatisierte Codegenerierung [JCL11]. Beispiele mit einem vergleichbaren Schwerpunkt von CPSen sind die Ansätze von Ringert et al. [RRW14] und Eidson et al. [ELM+12]. Dieser Arbeit entsprechend modelliert Ringert et al. CPSe über miteinander verknüpften Komponenten. Diese Komponenten besitzen Zustandsautomaten als eine interne Architektur und können über das *Eclipse Modeling Framework* grafisch modelliert werden. Sie ermöglichen eine formale Verifizierung und können über einen Codegenerator implementiert werden. Eidson et al. verwenden Programmiermodelle, die explizit die Verteilung von CPSen berücksichtigt. Mit diesen Modellen wird die Orchestrierung von Komponenten in CPSen modelliert, um über eine Codegenerierung deterministische, verteilte Echtzeitsemantiken bereitzustellen. Diese Orchestrierung von Komponenten in CPSen wird in dieser Arbeit nicht für Echtzeitsemantike, aber dafür für eine Evolutionsunterstützung genutzt. Dabei werden genauso Komponenten mit einer internen Architektur verwendet.

Im Allgemeinen werden in modellbasierten und -getriebenen Ansätzen Modelle für höherwertige domänenspezifische Entwicklungsmethoden, zur Codegenerierung sowie zur Analyse und Verifikation verwendet [JCL11]. Dabei können Entwicklungsmodelle in der Evolution insbesondere die Analyse und Bewertung von Änderungen unterstützen [Vya13]. Entsprechend werden Modelle auch in dieser Arbeit zur Abbildung des Verhaltens, der Architektur und

ihrer Eigenschaften eingesetzt. Allerdings verwenden die modellbasierten und -getriebenen Ansätze in der Regel manuell modellierte Modelle [Lad18]. Dazu werden Annahmen für nicht oder nur teilweise bekannten Aspekte getroffen und es werden die physischen Aspekte eines CPSs oft nicht vollständig modelliert. Deshalb werden, entgegen der vorgestellten Ansätze in dieser Arbeit, Modelle zur Laufzeit integriert und auf Basis von gemeinsamen Ereignissen der physischen und virtuellen Komponenten eines CPSs generiert. Dies entspricht nach der Kategorisierung von Shi et al. [SWYS11] einer Verwendung von Ereignismodellen für CPSe.

5.2.2 Regressives Testen

Testfälle repräsentieren ausführbare Modelle des Systemverhaltens und ermöglichen es, dieses Verhalten bezüglich der jeweiligen Ein- und Ausgaben zu überprüfen [RUPVH15]. Somit enthalten sie implizit Wissen über das Systemverhalten, welches durch ihre Ausführung erfassbar wird. Entsprechend sind Testfälle eine geeignete Methode, um Evolutionswissen abzubilden [MvZB08].

Dabei geht eine Vielzahl von Ansätzen beim Testen modellbasiert vor. Klassische Ansätze des *modellbasierten Testens* generieren aus einem Testmodell eine Vielzahl von Testfällen, die das im Modell spezifizierte Verhalten vollständig abdecken [AD97]. Ergänzend können mithilfe einer Auswirkungsanalyse (*impact analysis*) mögliche Effekte auf den Quellcode oder andere Artefakte bestimmt werden [LR+13]. Ansätze im Bereich von CPSen generieren Testfälle beispielsweise auf Basis von Automaten, Markow-Ketten, Petri-Netze oder UML-Diagrammen (siehe [KHMD12, HKVH+13])

Regressionstests beschreiben vormals korrekte Systemverhalten als Testfälle und jede neue Version bzw. Änderung wird gegen dieses Verhalten getestet. Für die Arbeit von besonderer Bedeutung sind Regressionstests des Verhaltens. Diese Art von Regressionstests nutzen zumeist die Software direkt, um Testfälle zu generieren und somit das momentan implementierte Systemverhalten abzubilden. Ein Beispiel, das Änderungen in den Mittelpunkt der Betrachtung stellt, entwickeln Jin et al. [JOX10] einen Ansatz, der zuerst Testfälle für den geänderten Teil des Quellcodes generiert und diese Testfälle dann auf dem ursprünglichen und den evolvierten Quellcode ausführt. Anschließend werden die unterschiedlichen Ergebnisse bestimmt und die daraus resultierenden Unterschiede dem Entwickler präsentiert. Unterschiede durch den Vergleich von Versionen werden auch für das Konzept der Evolutionsschritte in dieser Arbeit betrachtet. Dabei werden allerdings Änderungen aus dem beobachtbaren Verhalten und nicht aus dem Quellcode und entsprechenden Testfällen abgeleitet.

Diese Fokussierung auf Quellcode gilt auch für andere Ansätze von Regressionstests, da diese zumeist auf die Unterstützung der Entwicklungsphasen abzielen. Ein weiterer vergleichbarer Ansatz von Xie et al. [XTKM07] nutzt beispielsweise Vor- und Nachbedingungen von Methoden, um zwei Testfälle einer Klasse synchronisiert auszuführen und das resultierende Verhalten zu vergleichen. Evans et al. [ES07] generieren Testfälle aus zwei Versionen, um diesen über Behauptungen (*assertions*) miteinander zu verknüpfen. Dieses Vorgehen ist vergleichbar zur Erkennung von Abweichungen in dieser Arbeit, wobei die Erkennung nicht auf Testfällen, sondern auf Ereignismodellen durchgeführt wird. Dabei zeigen Li et al. [LXB+16], dass Ergebnisse von Testfällen auch zur Bestimmung von Ähnlichkeiten zwischen zwei Versionen verwendet werden können. Eine solches Ähnlichkeitsmaß wird in dieser Arbeit für Varianten realisiert.

Testverfahren wurden bereits auf unterschiedliche Anwendungsgebiete von CPSen übertragen. Beispielsweise haben Winkler et al. [WBÖ09] eine testgetriebene Entwicklung von CPPSen vorgestellt oder Hametner et al. [HWÖ+10] bestimmen Prioritäten für modellbasierte Testfälle von CPSen. Generell werden für Regressionstests Verfahren vorgeschlagen, die in

ähnlicher Form auch in der Evolution und für Laufzeitmodellen einen Mehrwert bieten. Dies gilt insbesondere für den wiederkehrenden Vergleich und die Berechnung von Unterschieden zwischen verschiedenen Versionen. Dies wird in dieser Arbeit mithilfe eines stetigen Prozesses zur Koevolution und zur Berechnung von Modelldifferenzen vorgenommen. Dies umfasst neben dem Verhalten auch weitere Aspekte einer Änderung, wie Gründe oder Änderungen von Eigenschaften.

Eine den Regressionstests konträrer Ansatz wird von Werner et al. [WG11] im Rahmen einer *Modellrekonstruktion* vorgeschlagen. Diese Autoren schlagen vor, Lerntechniken zur Extraktion eines Verhaltensmodells aus bestehenden Testfällen einsetzten. Dadurch wird implizites Wissen der Testfälle in explizites Wissen übertragen. Das so extrahierte formale Modell wird dann verwendet, um das laufende System zu überwachen und darüber sicherzustellen, dass das Systemverhalten mit seinen Testfällen übereinstimmt. Vergleichbare Ansätze stellen auch Torens et al. [TEL11] unter dem Begriff des *inversen modellbasierten Testens* für Verhaltensmodelle vor oder Ackermann et al. [ACH+10], die auf Extraktion von Anforderungen abzielen. Allerdings basieren auch diese Ansätze auf manuell spezifizierten Testfällen und es wird dabei nur sehr selektiv Bezug zu weiteren Aspekten von Änderungen genommen. Dennoch wird die unterliegende Idee als Erweiterung auch in der vorliegenden Arbeit aufgegriffen indem vorhandene Testfälle über eine Testfallausführung als Informationsquelle für die Koevolution eingesetzt werden.

5.2.3 Methoden des Reengineerings

Um Verhalten aus einem operativen System zu erfassen, können weiterhin auch Methoden des *Reengineerings* verwendet werden. Dabei haben dieses und angrenzende Forschungsfelder viele verschiedene Schwerpunkte und Anwendungsbereiche. Der Beitrag dieser Arbeit liegt diesbezüglich in der Einbindung solcher Methoden in ein übergeordnetes Konzept für Evolutionsschritte und die dadurch ermöglichte Unterstützung von Evolution. Die Arbeit adaptiert dafür ereignisbasierte Verfahren, da Ereignisse und ihre während der Ausführung gezeigten Muster wesentliche Eigenschaften eines operativen Systems beschreiben [SWYS11].

Aus dem Bereich des Reengineerings werden dazu insbesondere Methoden der Extraktion von Modellen (*modell recovery*) tangiert. Für Architekturmodelle stellt Ducaasse et al. [DP09] eine Kategorisierung vor. Der Schwerpunkt der vorliegenden Arbeit liegt aber auf einem einheitlichen Evolutionskonzept, weshalb diese Aspekte aller durch Ducasse und Pollet identifizierter Ziele miteinander verbindet. Diese Ziele umfassen die Dokumentation, die Analyse, die Koevolution und die Fortführung von Evolution [DP09]. Nachfolgend werden zunächst Ansätze mit vergleichbaren Zielen der Dokumentation, der Analyse und der Koevolution vorgestellt. Der daran anschließende Abschnitt widmet sich der Fortführung von Evolution.

Zur *Dokumentation* schlagen u.a. Ebert et al. [EB10, ERW08] und Nam et al. [NLM18] Softwaretools vor, die den Nutzer beim Verständnis der Evolution des Softwarecodes und seiner Systemarchitektur unterstützen. Zur Dokumentation wird dabei die Systemarchitektur auf einem dem Nutzer vertrauten Abstraktionsniveau von Modellgraphen visualisiert. Dazu werden Informationen der evolutionären Entwicklung des Quellcodes als Graphen in ein Repository extrahiert, das dann vom Nutzer betrachtet und abgefragt werden kann. Das Ziel der Evolutionsunterstützung dieser Arbeit ist es, ebenso das Verständnis von Evolution zu verbessern, allerdings indem Änderung des beobachtbaren Verhaltens in Modellen dokumentiert werden und darauf basierend dann passende Evolutionsschritte vorgeschlagen werden. Bei der Dokumentation wird weiterhin von vielen Autoren die Begründung (*rationale*) für die Architektur als wesentliches Kriterium untersucht [DP09]. So konzentrieren sich Durdik und Reussner [DR13]

beispielsweise auf die Dokumentation von architektonischen Gründen und Entscheidungen, die in Teilen auch extrahiert werden können (siehe Langhammer et al. [LSMR16]). Der Begriff des Grunds einer Evolution ist in dieser Arbeit allerdings anders belegt, denn damit ist hier nicht die Entscheidungsgrundlage des Nutzers gemeint, sondern das ungewollte Verhalten, das der Evolution vorausging.

Mit der *Analyse* eines extrahierten Modells können Eigenschaften des betrachteten Systems bestimmt werden, um den Nutzer in seinem Entscheidungsprozess zu unterstützen [DP09]. Unter dem Begriff der *qualitätsgetriebenen Extraktion* von Softwarearchitekturen zeigen Stormer et al. [SLV03] wie Metriken für Threads, Wartepunkte und weitere Eigenschaften der Performanz aus Architekturmodellen bestimmt werden können. Dieser Ansatz exportiert, wie viele vergleichbare Ansätze (siehe [MSC+18, TK18, EKS09]) auch Information in dedizierte Analysetools [DP09]. Einen interessanten Ansatz bezüglich Evolution verfolgt dabei das *ArchView-Tool* von Pinzger et al. [PGF08], welches über Repositorys für Quellcode abstrakte Metriken von Versionen bestimmt, um für die Evolution interessante Architekturelemente und Verbindungen zu identifizieren. Dies ermöglicht eine Evolutionsanalyse von Versionen, wobei eine Evolutionsanalyse im Sinne der Autoren eine Betrachtung der Gründe und Auswirkungen einer Änderung des Entwicklers ist [PGF08]. Diese Analyse ist grundsätzlich vergleichbar mit der in dieser Arbeit verfolgten Bestimmung von Eigenschaften. Dabei werden allerdings generierte Modelle verwendet und es wird ein zusätzlicher Schwerpunkt auf die Betrachtung von Eigenschaftsveränderungen über Modellversionen hinweg gelegt. Eine zusätzliche regelbasierte Untersuchung von Änderungsauswirkungen zwischen mehreren heterogenen Artefakten, wie dem Quellcode, Modellen und Testfällen, zeigen Lehnert et al [LR+13]. Ein solches Propagieren bei der Auswirkungsanalyse wird von der Arbeit bisher nicht verfolgt und wird als Teil des Ausblicks in Abschnitt 7.3 aufgegriffen.

Bezüglich der *Koevolution* werden eine Vielzahl von unterschiedlichen Zielen verfolgt. Dabei kann zur Verhinderung der als Problem identifizierten Wissenslücke nach Silva et al. [dSB12] grundsätzlich zwischen der *Minimierung*, *Verhinderung* und *Reparatur* einer Erosion von Softwarearchitekturen unterschiedenen werden. Methodisch lässt sich dabei die hier vorliegende Arbeit der *Reparatur* zuordnen. Allerdings wird diese - entgegen gängiger etablierter Ansätzen - direkt beim Auftreten einer Änderung eingeleitet und es wird zusätzlich die Fortführung von Evolution unterstützt. Diese Unterstützung ist nach der Kategorisierung von Silva et al. als eine Vorstufe zur *Selbstadaption* und damit der *Minimierung* zuzuordnen.

Für die Reparatur zeigen Tran und Holt [TH99] in frühen Arbeiten zur Koevolution bereits, dass extrahierte Abweichungen sowohl im Modell als auch im Quellcode durch eine Koevolution repariert werden können. Diesbezüglich existieren eine große Zahl an Methoden die, wie bei Testfällen, hauptsächlich den Quellcode aber auch dynamische Informationen verwendet. Für die vorliegende Arbeit sind insbesondere die Ansätze von Huang et al. [HMY06], Niere et al. [NSW+02] und Pinzger und Gall [PG02] von Interesse. Huang et al. [HMY06] analysieren den Laufzeitzustand über eine Laufzeitinfrastruktur, die es erlaubt die dynamische Architektur über Meta-Objekte, die Systemobjekten widerspiegeln, zu bewerten. Ein von der Idee vergleichbares Konzept wird auch in dieser Arbeit für die operative Verknüpfung von Modellen und dem CPS entwickelt. Einen musterbasierten Ansatz entwickelten Niere et al. [NSW+02]. Diese nutzen, wie auch die Evolutionsschritte dieser Arbeit, formale Transformationsregeln, die auf einem abstrakten Syntaxgraphen definiert werden. Diese werden allerdings nicht zur Beschreibung von Änderungen verwendet, sondern dienen der Wiederherstellung von Entwurfsmustern. Pinzger und Gall [PG02] verwendet String-Matching-Verfahren um vergleichbare Elemente einer Evolution zu identifizieren. Solche Verfahren werden in dieser Arbeit genauso bei der Zuordnung von Modellen für CBCPSe verwendet.

Morin et al. [MBJ+09] verfolgen einen zur Arbeit vergleichbaren Abgleich zwischen operativem und dokumentiertem Verhalten. Sie verwenden Sequenzdiagramme der Entwicklung, um ein laufendes System gegenüber seiner Spezifikation zu verifizieren. Allerdings wird dabei, entgegen dieser Arbeit, keine Koevolution angestrebt. Einen Schritt weiter gehen diesbezüglich Laufzeitmodelle, wie sie von Ivanovic et al. [ICH11] und Confora et al [CDEV08] vorgeschlagen werden. Diese Laufzeitmodelle aktualisieren Parameter, wie z.B. die Arbeitslast (*workload*) eines Systems, um beispielsweise die Performanz von operativen Systemen anhand von angereicherten Modellen zu analysieren. Pitakrat et al. [POvG18] zeigen in einem aktuelleren Ansatz weitergehend, dass auch die Anreicherung mit Laufzeitinformationen für mehrere miteinander verknüpfte Modelle möglich ist. In ihrem Ansatz kombinierenden die Autoren Architekturmodellen für Komponenten mit entsprechenden Fehlermodelle, um eine Vorhersage von Fehlern zu erreichen. Zwar werden in dieser Arbeit keine verschiedenen Typen von Modellen verknüpft, sondern Varianten über Laufzeitinformationen aktualisiert und zueinander in Beziehung gesetzt, um eine Aussage über mögliches zukünftiges Verhalten zu treffen.

Heinrich et al. [HJZ+17, Hei16] stellen einen vergleichbaren Ansatz vor, der eine Koevolution für Architekturmodelle von cloud-basierten Anwendungen anstrebt. Für ein Entwicklungsmodell wird eine *MAPE-K*-Feedback-Schleife für Laufzeitänderungen etabliert. Dabei werden vorhandene Entwicklungsmodelle in einer Systemarchitektur mit Laufzeitinformationen verknüpft und darüber aktualisiert. Dadurch werden Abweichungen und Engpässe erkannt und diese über den Nutzer für unterstützende sowie adaptierende Evolutionsaktivitäten bereitgestellt. So wird, genauso wie in dieser Arbeit, eine Koevolution von Modellen realisiert. Dabei fokussiert der Ansatz von Heinrich et al. die Verknüpfung von vorhandenen Entwicklungsmodellen sowie die Codegenerierung und Selbstadaption in einem Megamodell. In der vorliegenden Arbeit wird hingegen eine Generierung von vorher nicht bekannten Modellen und die Ableitung von modellbasierten Vorschlägen aus ähnlichen Systemen anstrebt.

Während ein Hauptziel vieler dieser Arbeiten ist, den menschlichen Nutzers zu eliminieren, kommen Heinrich et al [HJZ+17] genauso wie diese Arbeit zum Ergebnis, dass zunächst das Verständnis und die Unterstützung des Nutzers durch modellbasierte Ansätze für eine geordnete Evolution zielführender ist. Dabei handelt es sich bei den beiden Arbeiten um zwei der wenigen Arbeiten der Softwaretechnik, welche für die Evolution explizit eine Kombination von automatisierten Verfahren und einem menschlichen Nutzer als *human-in the-loop* anstreben (vergleiche [HJZ+17]).

Neben der Evolution existiert noch eine Reihe weiterer ähnlichen Ziele für die Unterstützung des Nutzers. Diese können unter dem Begriff der *Vorschlagssysteme* subsumiert werden (vergleiche [RWZ10]) und gewinnen ebenso Wissen aus u.a. *Issue Trackern* oder dem Quellcode. Dabei wird der Nutzer bei Softwareanpassungen des Quellcodes (siehe [LKKS15]), der Migration von Systemen (siehe [FH11]), der Optimierung von Eigenschaften (siehe [JMK+13]) oder der Visualisierung von operativen Daten (siehe [FRH15] durch Vorschläge unterstützt. Die vorliegende Arbeit verfolgt methodisch auch eine Unterstützung des Nutzers, grenzt sich allerdings durch eine anderes Ziel und andere technische Realisierung gegenüber den genannten Ansätzen ab.

5.2.4 Modellextraktion in CPSen

Neben vergleichbaren Zielen werden auch konkrete Ansätze für Einzelproblematiken in die, in dieser Arbeit vorgeschlagene, Koevolution eingebunden. Dies ist insbesondere für die Extraktion von Modellen der Fall. Dabei sind generische Verfahren (siehe [van11, LAD+11, GJL10, MCLM05]) für spezifische Modelle, wie in dieser Arbeit verwendet, in der Regel ungeeignet [HNS03, Lad18]. Deshalb werden nachfolgend nur verwandte Arbeiten der Extraktion für

die Modelle in den Anwendungsgebieten diskutiert und es werden die entsprechend dabei verwendeten Verfahren genannt.

Für CPPSe gibt es verschiedene Methoden auf Basis von statischen und dynamischen Informationen. Als statische Extraktionsmethoden werden zumeist modellabhängige Verfahren verwendet. Als ein Beispiel verwenden Rachetti et al. [RTF15] Softwarecode, um eine Dokumentation von speicherprogrammierbaren Steuerungen abzuleiten.

Allerdings sind für die Zielsetzung dieser Arbeit Methoden mit dynamischen Informationsquellen notwendig. Dies hat den Grund in der Dynamik des Anwendungsgebiets von Softwareevolution, der Interdisziplinarität von CPSen, die an der Schnittstelle zwischen Software und Physik beobachtet werden kann, und dem avisierten nichtinvasiven Beobachtungsansatz. Einige Verfahren, wie die von Mariani et al. [MPP11] und Lorenzoli et al. [LMP06], nutzen dafür interne Variablen oder Parameter der Software, was allerdings nur das Softwareverhalten widerspiegelt und andere Disziplinen von CPSen nicht berücksichtigt. Deshalb werden in dieser Arbeit Verfahren für operative und beobachtbare Ereignisfolgen verwendet, die das gesamte Verhalten eines CPSs abbilden. Diese Verfahren werden in CPPSen hauptsächlich zur Fehlererkennung eingesetzt und extrahieren Zustandsautomaten (siehe [BV17, PSG14, RLL10]), probabilistische Modelle (siehe [HA05]) oder Petri-Netze (siehe [LL11, LFA+15, LHFL15]).

Grundsätzlich können alle Arten von Ereignismodellen in das vorgeschlagene Konzept zur Evolutionsunterstützung integriert werden. Dabei werden in der vorliegenden Arbeit Petri-Netze nach Ladiges et al. [LFA+15, LHFL15] verwendet. Diese wurden im Rahmen, der dieser Arbeit zugrunde liegenden Forschungstätigkeit explizit für die Evolution des Materialflusses und von Maschinenzuständen entwickelt. Die Verfahren nutzten die Informationen an der Schnittstelle zu der speicherprogrammierbaren Steuerung, um Verhalten zu extrahieren und Eigenschaften aus den Modellen abzuleiten.

Für CBCPSe sollen explizit etablierte, nicht für die Evolutionsunterstützung spezifisch entwickelte Modelle verwendet werden. Die Arbeit erfordert dabei, wie auch bei CPPSen die Definition in einem Meta-Modell, die Möglichkeit zur Extraktion aus Laufzeitinformationen, ein Verfahren zur Bestimmung von (nichtfunktionalen) Eigenschaften sowie für die prototypische Evaluation, die Einbindung in Komponenten einer *Java*-Laufzeitumgebung. Unter diesen Rahmenbedingungen gibt es eine Vielzahl von möglichen Modelltypen. Standards sind beispielsweise die *Software Measurement Modeling Language* [MGRP09] oder Modelle auf Basis des *Structured Metrics Meta-Modells* [Obj12]. Diese stellen statistische Architektur- und Codemetriken bereit, bieten allerdings keine direkte Verbindung zur Modellextraktion oder zur Einbindung in eine Laufzeitumgebung [Wal18].

In dieser Arbeit wird das PCM-Werkzeug (*Palladio Component Model*) [RBB+] mit seinen Modellen u.a. für CBCPSe verwendet. Denn dabei handelt es sich um in der Forschung etablierte Modelle von komponentenbasierten Systemen, für die in der Literatur bereits Verfahren zur Extraktion aus Ereignissen vorliegen (siehe [DP09]). Das in dieser Arbeit entwickelte Verfahren nutzt methodische Ergebnisse der Arbeiten von Brosig et al. [BKK09] und Krogmann [Kro12]. Borsig [BKK09] stellt ein Verfahren zur Generierung von PCM-Modellen aus einer JavaEE-Anwendung vor. Er verwendet dafür Messdaten, die mithilfe eines externen Beobachtungswerkzeuges zur Laufzeit erhoben werden. Krogmann [Kro12] stellt ein erweitertes Verfahren für die Kombination von dynamischen und statischen Informationen vor. Dadurch können die statische Systemarchitektur eines komponentenbasierten Softwaresystems und das dynamische Verhalten von Diensten extrahiert werden.

Der Schwerpunkt dieser Arbeit liegt allerdings nicht auf dem Verfahren der Extraktion, sondern auf einem einheitlichen Konzept von verschiedenen Aspekten eines Evolutionsschrittes. Deshalb wurde ein spezifisch auf diese Bedürfnisse zugeschnittenes Verfahren für Systeme aus *Aktiven Komponenten* entwickelt (siehe [Dad17]). Dieses setzt Teile der von Brosig et al. und Krogmann vorgeschlagenen Verfahren für spezifische Codeelemente von *Aktiven Komponenten* um. Die Methode beschränkt sich dabei auf die Extraktion einer statischen Komponentenarchitektur und auf *Grey-Box*-Verhaltensmodellen von Methoden aus dynamischen Messwerten.

Entsprechend existieren in der Literatur Alternativen und Erweiterungen, die über das realisierte Verfahren dieser Arbeit hinausgehen. Als ein Beispiel schlägt Walter [Wal18] eine beschreibende Sprache für performanzrelevante Fragestellungen, eine Entscheidungsunterstützung für Leistungsbewertungsansätze und eine Schätzung der Analysezeit für die modellbasierte Leistungsvorhersage vor. Der Ausblick in Abschnitt 7.3 zeigt, dass diese Erweiterungen auch in der hier vorgestellten Evolutionsunterstützung Anwendung finden können.

5.3 Evolutionsunterstützung in CPSen

Als dritter Forschungsgegenstand wird eine weitergehende Evolutionsunterstützung des Nutzers in Netzwerken untersucht.

5.3.1 Themenfeld der Evolutionsunterstützung

Als ein wesentlicher Aspekt der Evolution identifizieren Buckley et al [BMZ+05] bei deren Beschreibung und Ducasse und Pollet [DP09] bei der Rekonstruktion von Modellen die Fortführung der Evolution. Die vorliegende Arbeit verfolgt dieses Ziel indem das Netzwerk eines CPSs genutzt wird, um einen Austausch von modellbasierten Modellen zu realisieren und aus diesen Vorschlägen zur Fortführung der Evolution abzuleiten.

In CPSen wird ein Austausch meist in klassischen Anwendungsbereichen, wie z.B. den Produktionsleitsystemen und der Ressourcenplanung, im Sinne eines Teilen von Ressourcen betrachtet (vergleiche [Vya13]). Für CPPSe untersuchen beispielsweise Freitag et al. [FBD15] und Valilai et al. [VH13] die Dynamik eines Teilen bzw. einer Virtualisierung von Maschinenressourcen in Produktionsnetzwerken. Diese Ansätze fokussieren auf die Kooperation bezüglich der verfügbaren Ressourcen. Der für die vorliegende Arbeit gewählte Ansatz betrachtet allerdings das Teilenvon Informationserfahrungen und verbindet CPSe somit mit dem Wissens- oder Erfahrungsmanagement (siehe [Wil13]).

Die dafür grundlegenden Methoden und die Systemarchitektur für einen Austausch von Evolutionswissen beschreiben Lee et al. [LBK15]. Die Autoren entwerfen in ihrem viel beachteten Artikel eine Vision einer *Zeitmaschine* für selbstwahrnehmende CPSe, die in wesentlichen Elementen auf diese Arbeit übertragbar ist. Dabei sollen sich Maschinen in einem Netzwerk registrieren, um Informationen auszutauschen. Das Netzwerk, welches in dieser Arbeit dezentral über ein Peer-to-Peer-Netzwerk implementiert ist, realisiert eine Bestimmung und Nachverfolgung von Änderungen. Dafür ist eine Sammlung von Zuständen (*snapshots*) notwendig. Diese wird in dieser Arbeit durch die Verknüpfung von Modellen und Laufzeitinformationen sowie die Implementierung von Evolutionsschritten realisiert. Weiterhin muss nach Lee et al. das Netzwerk das Wissen auf Ähnlichkeit prüfen, was von der Idee der in dieser Arbeit vorgeschlagenen Ableitung von Modelldifferenzen entspricht. Als letzte Aktivität nennen Lee et al. eine Synthese von zukünftigen Aktivitäten, welche in dieser Arbeit mit einer Unterstützung durch Vorschläge realisiert ist. Auch wenn die Zukunftsvision von Lee et al. allgemeiner ist und

über die Evolutionsunterstützung hinausgeht, kann die in dieser Arbeit entworfene Methode und Architektur als eine Instanz einer solche Zeitmaschine verstanden werden.

Vergleichbare Arbeiten, die ebenfalls zu dieser Vision beitragen, existieren insbesondere im Bereich der Selbstwahrnehmung. Als ein Beispiel stellen Kwon et al. [KHF+16] einen Ansatz vor, wie Methoden der Prognostik und des Gesundheitsmanagements auf das Internet der Dinge übertragen werden können. Die Autoren konzentrieren sich allerdings auf spezifische Methoden zur Erkennung von Fehlern und nicht die Evolutionsunterstützung. Dennoch verfolgen diese Arbeiten den gleichen Ansatz eines Vergleichs von aktuellem und gewolltem Systemverhalten eines CPSs. Zhang et al. [ZQLL17] entwerfen Modelle für CPSe, die eine Selbstorganisation und Selbstadaption auf Basis von Agenten ermöglichen. Der Ansatz geht allerdings, wie vergleichbare Ansätze auch, von einer Top-Down-Betrachtung aus. Diese Arbeit verwendet hingegen einen Bottom-Up-Ansatz, um Schlussfolgerungen für die Evolution abzuleiten. Eine Überschneidung besteht deshalb nur bei der Erstellung von Modellen als Vorschlag für die Fortführung der Evolution - wobei diese Arbeit nur von beobachtetem Verhalten ausgeht und deshalb nur die modellbasierte Unterstützung zum Ziel hat.

Ein Austausch von Wissen zur modellbasierten Unterstützung von ausführbaren, evolutionären Änderungen über Netzwerke von horizontal integrierten Systemen wurde bisher nicht verfolgt. Allerdings wird in einem softwaretechnisch vergleichbaren Ansatz durch Boschian et al. [BDF+11] ein Netzwerk entwickelt, welches auf Informationen in einer Wissensbasis zurückgreift und dadurch Systemverhalten vorherzusagen versucht. Eine solche Vorhersage wird in dieser Arbeit auch angestrebt, wobei bereits durchgeführte Modelländerungen verwendet und neben der Verhaltensänderungen auch weitere Aspekte, wie der Grund und die Auswirkungen auf Eigenschaften, betrachtet werden.

Es existieren weitere angrenzender Arbeiten für spezifische Aspekten eines Austausches. Als ein interessantes Beispiel dafür untersuchen Jiang et al. [JDL16] den sozialen Aspekt eines Austausches. Dabei schlagen die Autoren ein dienstorientiertes Wissensnetzwerk im Sinne eines sozialen Netzwerks wie Facebook vor. Dies entspricht in Teilen dem kooperativen Ansatz dieser Arbeit, der softwaretechnisch durch einen Marktplatz für Evolutionsschritte realisiert wird.

Zusätzliche domänenspezifische Ansätze sind insbesondere für CPPSe vorhanden. Unter dem Ziel der Fortführung von Evolution betrachten einige Autoren u.a. wiederverwendbare Module. Diese werden verwendet, um Änderungen aus vorangegangenen Systemen auf zukünftige Systeme zu übertragen. Diesbezüglich wird auch die Koevolution mithilfe von Änderungen in der *Automation Markup Language* (siehe Berardinelli et al. [BBM+15]) oder mit *plug & produce*-Systemen (siehe Schleipen et al. [SLS+15]) betrachtet. Diese Ansätze versuchen eine Verbesserung von zukünftigen, interdisziplinären Systemen durch die Übertragung von Erfahrungen aus der Evolution zu erreichen. Ein automatisierter, kooperativer Austausch von beobachteten Änderungen zur Laufzeit wird hier allerdings nicht verfolgt.

Bezüglich einer kooperativen Unterstützung durch Wissen der Evolution stellen Würsch et al. [WGG13] ein System vor, welches ein anfragebasiertes Netzwerk von Evolutionswissen für Entwicklungswerkzeuge zum Ziel hat. Durch dieses Netzwerk sollen die bereits vorliegenden Erfahrungen einer Evolution erhalten bleiben. Das System fokussiert dabei allerdings nicht auf Erfahrungen des laufenden Systems selbst, sondern basiert auf abgeleiteten Informationen von Repositorys der Entwicklung.

Aus einer methodischen Sichtweise stellt die Delta-Modellierung von Softwareproduktlinien eine Methode dar, welche bei der Repräsentation und Anwendung vergleichbar mit dem vorgeschlagenen Konzept von Evolutionsschritten dieser Arbeit ist. Bei der Delta-Modellierung wird

ein System in ein Kernmodul und eine Menge von Delta-Modulen unterteilt, die inkrementell zusammengefügt werden [SBB+10]. Als ein vergleichbarer Ansatz aus diesem Bereich führen Seidl et al. [SSA14] das Werkzeug *DeltaEcore* ein. Dieses ermöglicht es, aus einer vorhandenen Modellierungssprache eine Delta-Modellierungssprache abzuleiten. Dazu werden zunächst Syntax und Semantik des Meta-Modells der Modellierungssprache analysiert und dann inkrementelle Sprachmodule abgeleitet. Im Konzept der vorliegenden Arbeit werden ebenso inkrementelle Änderungen betrachtet, allerdings wird dabei auf die Ebene von Modellinstanzen zur Laufzeit fokussiert und nicht, wie bei den Sprachmodulen, auf die Ebene des Meta-Modells der Entwicklung.

5.3.2 Ansätze für Modelldifferenzen

Eine wesentliche softwaretechnische Fragestellung ist die, wie Änderungen zwischen zwei Modellversionen softwaretechnisch operationalisiert und abgebildet werden können [CWU+18]. In unterschiedlichen Anwendungsbereichen von CPSen wurden Änderungen dafür in den letzten Jahren beispielsweise über verschiedene Entwicklungsphasen und Disziplinen hinweg miteinander verknüpft (siehe Regulin et al. [RAVH16]) oder bezüglich ihres Einflusses bewertet (siehe Feldmann et al. [FKVH14]). Wie die genannten Ansätze auch, verfolgen Regulin et al. [RAVH16] in diesem Zusammenhang explizit die Integration von Meta-Modellen in den Lebenszyklus eines CPSs, um die Übertragbarkeit und Konsistenz zu verbessern. Allerdings werden in diesen Ansätzen grundsätzlich Entwicklungsmodelle verwendet und die Bewertung und Evolution über die Versionen hinweg wird dabei vernachlässigt [PKH+18].

Bei der Betrachtung von Versionen bietet die modellbasierte Forschung jedoch bereits passende Methoden und Versionierungswerkzeuge an. Diese verbessern die Beschreibung und Aussagekraft über Differenzen von Modellversionen. Für die Beschreibung von Differenzen schlagen Brosch et al. [BSW+09] ein Zusammenführen von Editieroperationen vor. Dazu verwenden die Autoren Beispiele des Entwicklungsprozesses, um mögliche Editieroperationen von Modellen zu erfassen und diese zusammenzuführen. Einen ähnlichen Ansatz mit Beispielen verfolgen Langer et al. [LWB+13, Lan11], wobei insbesondere ein verbessertes Verständnis von Änderungen erreicht werden soll. Beide Ansätze betrachten allerdings nicht das operative Verhalten eines Systems oder wenden diese Ansätze auf Ereignisse von CPSen an. Nichtsdestotrotz sind sie vergleichbar zu der vorgeschlagenen Methode dieser Arbeit, da die Beschreibung von Änderungen auch auf Operationen von zusammengeführten Differenzen beruht. In dem Ansatz von Cicchetti et al. [CDP09] wird eine Anwendung von Differenzen durch eine Modelltransformation angestrebt. Dabei beschreibt eine Regel immer eine vollständige Transformation von einer Version in eine folgende Version des Modells. Ein solcher Ansatz ist für die avisierten Unterstützung über einen Marktplatz eher ungeeignet, da Änderungen damit nur schwer trennbar und abwandelbar werden. Ein flexiblerer Prozess wird durch Könemann [Kön10] vorgeschlagen, wobei dieser nur schwer eine Automatisierung ermöglicht.

Deshalb wird für die vorliegende Arbeit der Ansatz von Kehrer et al. [KKT11, KKOS12, KKT13, Keh15] für die hierbei verwendeten koevolvierenden Modelle und für das Konzept von Evolutionsschritten angepasst. Dieser Ansatz wurde bereits im Grundlagenkapitel erläutert; er ermöglicht es, semantisch geliftete Differenzen zu bilden und über Editierskripts auf dritte Modelle zu übertragen. Der Ansatz wurde bereits durch Kehrer et al. [KKT14] im Bereich der CPPSe angewendet. Dabei werden allerdings, entgegen dieser Arbeit, zur Entwicklungszeit manuell modellierte, strukturierte Blockdiagramme verwendet. Die Anwendung zeigt jedoch, dass die Methode auch für Modelle von CPSen geeignet ist.

5.3.3 Beobachtung von CPSen

Für die vorgeschlagene softwaretechnische Systemarchitektur gibt es eine Reihe verwandter Arbeiten. Dabei kann zunächst das Verhalten von CPSen durch unterschiedliche softwaretechnische Methoden erfasst werden. Generell existieren zur Instrumentalisierung und Erfassung von Ereignissen dieser Systeme eine Vielzahl von *Beobachtungswerkzeugen und Frameworks*. Als ein Beispiel stellen van Hoorn et al. [vRH+09] ein Framework zur generischen Beobachtung von Softwaresystemen vor. Dieses ermöglicht es, den internen Ablauf von Aufrufen oder auch verteilte Ereignisfolgen zu beobachten und zu verarbeiten. Dazu werden Anwendungen, wie ein CPS, mit so genannten Proben über ihren Quellcode erweitert und die Messdaten an Analyse- und Logging-Komponenten weitergereicht. Die vorliegende Arbeit verwendet für CPPSe die Erfassung von Ereignissen über Bussystem und für CBCPSe die bereits integrierten Beobachtungsverfahren von *Aktiven Komponenten*. Die resultierenden Ereignisse sind allerdings vergleichbar zu denen von generischen Frameworks.

Dabei bieten Frameworks bereits die Möglichkeit des Zusammenfügens von Ablaufverfolgung (*trace*). Dies betrifft in dieser Arbeit das Zusammenfügen von Ereignissen von CBCPSen. Für diese Aufgabe sind insbesondere die Arbeiten zur Rekonstruktion von Komponenteninteraktionen über Methodenaufrufe (siehe Israr et al. [IWF07]), das Zusammenführen von verteilten Webservice-Aufrufen (siehe Okanovic et al. [OvKV11] und die Extraktionen von externen Anfragen (siehe Barham et al. [BDIM04]) relevant. Als für die vorliegende Arbeit am besten passender Ansatz für Komponentenmodelle werden die Vorarbeiten zur zeitlichen Verhaltensanalyse und Fehlerlokalisation von Rohr [Roh15] genutzt. Diese ermöglichen das Bilden von Nachrichtensequenzen, was auf Ereignisse von *Aktiven Komponenten* übertragen wird. Diese Nachrichtensequenzen bilden die Grundlage der Extraktion von PCM-Modellen.

Im Umfeld von industriellen Anlagen sind weitergehen *bedingte Beobachtungsansätze* (conditional monitoring) als verwandte Arbeiten zu nennen. Einen Überblick dazu geben Ambhore et al. [AKCW15]. Diese Ansätze werden insbesondere für die Fehlererkennung und -diagnose eingesetzt [FKF15]. Fleischmann et al. [FKF15] stellen diesbezüglich einen Ansatz für CPSe vor, der dezentrale Beobachtungsmodelle verwendet, die beobachtete Daten über Cloud-Module verarbeiten. Dabei werden in diesem, wie in vergleichbaren Ansätzen der Fehlerdiagnostik, häufig agentenbasierte Systeme eingesetzt, die zumeist inhärent in den Prozess eingebettet sind [PPS+05]. Während in dieser Arbeit die Beobachtung über Modelle vergleichbar zu Fleischmann et al. realisiert wird, ist die Einbettung von Agenten in das beobachtete CPS nicht gewünscht, da die Ausführung des CPSs durch die Beobachtung unbeeinflusst bleiben soll.

Merdan et al. [MVLZ11] schlagen *Automationsagenten* vor, die spezifische Feldhardware beobachten und daraus ein Weltmodell eines Produktionssystems erstellen. Aus diesem können über Ontologien Schlussfolgerungen gezogen werden. Bezüglich der Beobachtung folgt deren Ansatz einer zu dieser Arbeit vergleichbaren Idee, bei dem Schlussfolgerungen entlang der Hierarchie von technischen Komponenten gezogen werden. Unterschiede bestehen dabei allerdings in der hardwarenahen Verarbeitung sowie in dem in dieser Arbeit verfolgten Verarbeitungsprozessen für die Evolution. Der Ansatz von Seilonen et al [SPH+06] verwendetet die gleiche BDI-Agentenarchitektur wie diese Arbeit. In deren Ansatz kann der Nutzer Beobachtungsaufträge erteilen, die unterschiedliche Prozesseigenschaften betreffen. Dabei werden Ziele und Pläne verwendet, um die Verarbeitung in Informationsagenten zu steuern. Diese Steuerung wird durch *Aktive Komponenten* auch umgesetzt, wobei in dieser Arbeit mit der Evolutionsunterstützung über Dienste von Modellen ein anderes Ziel verfolgt wird.

Eine daran anknüpfende Vorgehensweise bei CPSen ist die Verwendung von *digitalen Zwillingen* (*digital twin*). Diese koppeln eine Cyber-Komponente mit der physischen Komponente,

um ihr Wissen und ihre Bedingungen für weiterführende analytische Methoden vorzuhalten und verfügbar zu machen [LLBK13, SPT19]. Einen Überblick dieser Ansätze für CPSe geben Negri et al. [NFM17]. Dabei gibt es u.a. methodische Vergleichbarkeiten zu Arbeiten der *Wissensintegration* (siehe Novak et al. [NSMŠ15]), der *Rekonfiguration* (siehe Malec et al. [MNNN07]), der *Performanzvorhersage* [BHH15], der *semantischen Beschreibung* (siehe Hastbacka et al. [HK13]) und der *Big-Data*-Analyse (siehe Lee et al. [LLBK13]). Dabei hat die Arbeit mit diesen Ansätzen gemein, dass Wissensartefakte über den unterliegenden Prozess entlang der Hierarchie von technischen Komponenten gehalten werden. Unterschiede bestehen in der Aufrechterhaltung von Wissen, der in dieser Arbeit dem Prozess zur Koevolution folgt, und im Ziel der Evolutionsunterstützung durch ein Netzwerk, welches für digitale Zwillinge erstmals untersucht wird.

5.3.4 Selbstwahrnehmung von CPSen

Bei einer Wissensspeicherung über den Status, die Situation und die Handlungsoptionen wird von *selbstwahrnehmenden Systemen* gesprochen (vergleiche [ST09]). Diese und die daraus resultierenden Eigenschaften, wie die Selbstdokumentation, die Selbstadaption, die Selbstoptimierung oder die Selbstanalyse werden in der Softwaretechnik durch überlappende Forschungsbereichen untersucht [BG09].

Unter dem Begriff einer *modellbasierten Beobachtung* (*model-based monitoring*) werden hierzu u.a. Softwarearchitekturen vorgeschlagen, die durch koevolvierende Modelle eine Selbstadaption ermöglichen. Ähnlichkeit zu Konzepten dieser Arbeit haben dabei insbesondere die Ansätze von Garlan et al. [GCH+04, CdLL+14] und Le Duc et al. [LCMR10]. Garlan et al. nutzen Systeminformationen auf niedrigen Abstraktionsniveaus, um ein Adaptionsmodell zu erstellen, das über Beobachtung aktuell gehalten wird. Über dieses Adaptionsmodell können Strategien zur Adaption abgeleitet werden. In dieser Arbeit werden Modell ähnlich verwendet, wobei die Ableitung zur Unterstützung des Nutzers dient und auf der Bestimmung von Modelldifferenzen beruht. Eine wiederverwendbare Infrastruktur für die Evolutionsunterstützung wird dabei allerdings genauso wie die wiederverwendbare Infrastruktur für selbstadaptive Systeme von Garlan et al. verfolgt. Le Duc et al. [LCMR10] stellen ein generisches Framework für Bedingungen von dynamische Datenströmen vor. Das Framework basiert auf Modellen der Systemressourcen und ihrer Bedingungen und Anfragen. Es ermögliche die Spezifizierung von Bedingungen auf solche Datenströmen. Das Framework wurde dabei in unterschiedlichen Ansätzen weiterverwendet, was auch einen Ansatz für die Koevolution umfasst (siehe Song et al. [SHC+10]). Die vorliegende Arbeit verfolgt mit einem Konzept von strategieorientiert ausgewerteten Aussagen auch ein Konzept, das Bedingungen an Datenströme formuliert. Der Fokus dieser Arbeit liegt dagegen auf einem verbesserten Konzept zur Auswertung in einer gekapselten Ausführungsumgebung sowie auf einer effizienten Auswertung über Strategien.

5.3.5 Dienstorientierte CPSe

Bezüglich der verwendeten Softwareparadigmen dieser Arbeit werden auch *dienstorientierte CPSe* in der Literatur genannt. Dabei nennen Hoang et al. [HPK12] die Koordination von Software- und Hardwarekomponenten als hauptsächliches Entwurfsziel der Dienstorientierung. Die darin verfolgten Ansätze fokussieren in der Regel auf eine Middleware für CPSe. Beispielsweise beschreiben Wu et al. [WTRS12] eine semantische Middlware, die heterogenen CPS-Elemente miteinander verbindet und eine Orchestrierung ermöglicht. Valilai et al. [VH13] beschreiben die gemeinsame und kooperative Nutzung von Ressourcen, Candido et al. [CCBJ11]

die Komposition von Modulen während der Evolution und Lin et al. [LP10] fokussieren auf die Vorhersagen in CPSen. Eine Verknüpfung, kooperative Nutzung und Komposition sowie Vorhersage wird auch in dieser Arbeit verfolgt. Allerdings wird dies über autonome Dienstkomponenten für Wissensmodelle angestrebt. Entsprechend schlägt die vorliegende Arbeit eine verteilte Architekturtopologie als Kompromiss zwischen Vorteilen von Diensten, Agenten und Komponente vor, die mit der Verwaltung von Wissen nach der Kategorisierung für dienstorientierte CPSe von Hoang et al. [HPK12] einen sehr starken Fokus auf der Anwendungsebene hat.

5.4 Zusammenfassung

Zusammenfassend tangieren die in der vorliegenden Arbeit verwendeten, neu vorgeschlagenen oder weiter entwickelten Konzepte unterschiedliche Forschungsbereiche. Dabei werden verschiedene Methoden dieser Forschungsbereiche miteinander kombiniert und erweitert bzw.-mit neuen Methoden ergänzt, um letztendlich eine neuartige Art der Koevolution und Evolutionsunterstützung für CPSe zu erreichen. Diese können im Verhältnis zu den verwandten Arbeiten wie folgt zusammengefasst und eingeordnet werden:

Die beschriebenen Taxonomien werden durch eine aspektübergreifende Sichtweise auf Änderungen während der Evolution erweitert. Das dazu entwickelte Konzept operationalisiert die vorhandenen allgemeinen Taxonomien über eine Beschreibung mit Modellen, was für die Abweichungs- und Fehlererkennung in Teilen, aber für die Evolutionsunterstützung bisher noch nicht realisiert wurde (Abschnitt 5.1).

Die isoliert von der Evolution entwickelten Verfahren zur Generierung von Modellen, Erkennung von Abweichungen und Analyse von Eigenschaften werden in dieser Arbeit aufgegriffen und verwendet. Diese werden über Methoden von Laufzeitmodellen für eine Koevolution verknüpft. Dabei werden softwaretechnisch vorhandene agentenbasierte Arbeiten zur Steuerung der Koevolution und zur Auswertung von Ereignisströmen durch eigene Konzepte ergänzt (Abschnitt 5.3).

Unterschiedliche Softwareparadigmen werden bereits in verschiedenen vergleichbaren Ansätzen für CPSe verwendet und können auch für das neue Ziel der Evolutionsunterstützung genutzt werden Aufbauend auf diesen Softwareparadigmen für CPSe wird durch die vorliegende Arbeit ein Netzwerk für CPSe betrachtet, das eine bisher noch nicht erforschte Evolutionsunterstützung für den Nutzer durch Vergleiche von ähnlichen Varianten verfolgt. Dazu können zur Beschreibung von Unterschieden solcher Varianten u.a. Methoden zum semantischen Liften von Modelldifferenzen auf Ereignismodelle für CPSe übertragen werden. (Abschnitt 5.2).

Die Kombination von beschreibenden, modellbasierten Evolutionsschritten, die durch operationalisierte Aspekte zu einer vergleichenden Evolutionsunterstützung beitragen, stellt somit eine Komposition und Erweiterung vorhandener Methoden in unterschiedlichen Forschungsgebieten dar. Dabei verfolgt der in dieser vorliegenden Arbeit verfolgte Ansatz eine operationalisierte Beschreibung aller Aspekte einer Evolution über sequenzielle Schritte (vergleiche [BMZ+05]). Dies lässt sich nach den aufgezeigten Kategorisierungen in diesem Kapitel (siehe [BMZ+05, dSB12, DP09, SWYS11]) methodisch wie folgt klassifizieren: Es wird eine ständige *Beobachtung* von (gewollten und ungewollten) Änderungen und ggfs. eine *Reparatur* und teilweise damit eine *Minimierung* von Alterungsprozessen verfolgt (vergleiche [dSB12]). Dabei werden Konzepte für die *Koevolution* und eine *Fortführung* von Evolution (vergleiche [DP09]) sowie die *Analyse* und *Dokumentation* in *Ereignismodellen* für CPSe entwickelt (vergleiche [DP09, SWYS11]).

6 Experimentelle Evaluation der Evolutionsunterstützung

Im folgenden Kapitel wird die entworfene Methodik der Evolutionsunterstützung in der entwickelten Systemarchitektur anhand von zwei Fallstudien evaluiert. Dazu werden in Abschnitt 6.1 fünf Ziele der Evaluation aus den Forschungsfragen abgeleitet. Um entsprechende Experimente zur Evaluierung dieser einzelnen Ziele durchzuführen, werden in Abschnitt 6.2 zunächst mit einer *Pick-and-Place-Unit* und einem *Kassensystem eines Supermarktes* zwei Fallstudien für CPPSe und CBCPSe vorgestellt. In Abschnitt 6.3 werden dann die Machbarkeit und Ergebnisse dieser Experimente gezeigt und diskutiert.

6.1 Ziele der Evaluation

Die Hauptforschungsfrage dieser Arbeit beschäftigt sich mit einer systematischen Verwaltung von modellbasierten Evolutionsartefakten für CPSe. Aus einer allgemeinen Sichtweise wird damit die Frage einer geeigneten Beschreibungsform aufgeworfen. Diese wurde im Rahmen dieser Arbeit durch das auf spezifische CPSe anpassbare Konzept von Evolutionsschritten mit Aspekten nach dem CRI-Modell beantwortet. Dabei ist es entscheidend, ob Evolutionsschritte die Evolution von unterschiedlichen CPSen abbilden können. Entsprechend wird deshalb zunächst folgendes Ziel bei der Evaluation verfolgt:

- Die wesentlichen Aspekte von Evolutionsschritten sollen auf die Fallstudien abgebildet und die entsprechenden Abbildungen durch typische Evolutionsszenarien bewertet werden.

Als wesentliche Grundlage wurde in der temporären Forschungsfrage eine systematische Gewinnung von Wissen thematisiert, das operativ durch eine wissenstragende Komponente extrahiert wird. Aus der Problemstellung einer heterogenen Umgebung in CPSen ergibt sich so die Notwendigkeit einer Einbindung dieser heterogenen Quellen in die Evolutionsunterstützung. Zur Evaluation wird deshalb weiterhin das folgende Ziel verfolgt:

- Verschiedene Ereignisquellen für die Evolutionsunterstützung sollen in die wissenstragende Komponente eingebunden werden und der dadurch für die Evolutionsunterstützung erreichte Mehrwert bewertet werden.

Als wesentliches Kriterium für die Evolutionsunterstützung wurde eine Koevolution von Modellartefakten verfolgt. Dies resultiert aus der methodischen und operativen Forschungsfrage einer konsistenten Verwaltung von Evolutionsartefakten. Dazu wird im nächsten Ziel untersucht, ob eine Koevolution für verschiedene Typen von Modellen und CPSen möglich ist:

- Die entwickelten Modellartefakte sollen auf die Fallstudien angewendet werden, wobei jeweils ihre Generierung, Beobachtung und Analyse im Einzelnen bewertet wird.

Als wichtiges Konzept der Arbeit ist eine Verknüpfung in Evolutionsschritten vorgesehen. Dazu werden verschiedene Aspekte von Evolution untersucht und es wird die Frage gestellt, wie Modellversionen als Evolutionsschritten dokumentiert und zur Fortführung von Evolution genutzt werden können. Diese Unterstützung wird als Evaluationsziel wie folgt formuliert:

- Aus den verschiedenen Versionen von Modellartefakte der Fallstudie von CPPSen sollen Evolutionsschritte erstellt und zwischen wissenstragenden Komponenten ausgetauscht werden. Weiterhin soll der entstehende Evolutionsprozess betrachtet werden.

Neben dem primären Ziel der Koevolution und der Fortschreibung von Evolution in Versionen können Evolutionsschritte auch unter anderen Anwendungszielen und -kontexten genutzt werden. Bezüglich Varianten wird deshalb untersucht, inwieweit über Evolutionsschritte ein modellbasiertes Ähnlichkeitsmaß für Varianten definiert werden kann. Dieses weitergehende Ziel soll zusammen mit der Anwendung auf allgemeine Entwicklungsmodelle in einem zusätzlichen Experiment mit CBCPSen überprüft werden:

- Evolutionsschritte sollen genutzt werden, um zwei Varianten der Fallstudie von CBCPSen über ein Ähnlichkeitsmaß zu bewerten.

6.2 Fallstudien

Für die Experimente werden zwei unterschiedliche Fallstudien verwendet. Dies ist zum einen eine *Pick-and-Place-Unit (PPU)*, die als Fallstudie für CPPSe genutzt wird. Zum anderen wird eine Kassensystem für Supermärkte als Fallstudie für CBCPSe betrachtet.

Die Fallstudien stammen aus dem Schwerpunktprogrammes 1593 der deutschen Forschungsgemeinschaft[1] und wurden für unterschiedliche Methodiken der Evolutionsforschung entwickelt und eingesetzt. Die Fallstudien haben daher den Vorteil, dass sie unabhängig von der Problemstellung der Arbeit entwickelt wurden und eine realistische, unabhängige Testumgebung darstellen. Dabei versuchen die Fallstudien einen guten Kompromiss zwischen Problemkomplexität und Evaluationsaufwand zu finden [VHLFF]. Die cyber-physischen Aspekte sind dabei in den beiden Fallstudien unterschiedlich stark ausgeprägt.

Für die Evaluation werden beide Fallstudien zunächst in ihrem Aufbau erläutert. Die in den Experimenten notwendigen Szenarien und Komponenten werden dabei überblicksartig erläutert. Für weitere Details der Fallstudien wird auf entsprechende Veröffentlichungen verwiesen (siehe [VHLFF, HGH+15]).

6.2.1 PPU-Komponente als CPPS

Die erste Fallstudie entstammt dem Gebiet der Automatisierungstechnik. Die verwendete Pick-and-Place-Unit (PPU)[2] dient in dieser Arbeit zur Untersuchung von CPPSen. Die Fallstudie wird als Demonstrator für Forscher aus dem Bereich der Evolutionsforschung angeboten und bietet eine Vielzahl unterschiedlicher Evolutionsszenarien an (siehe [VHLFF]).

Eine PPU-Komponente wird durch eine SPS gesteuert, die mittels Ereignissen über ein Bussystem mit den angeschlossenen Automatisierungskomponenten kommuniziert. Diese Ereignisse stammen entweder von Sensoren (z.B. das Signal *Sensor.WorkpieceReady*) oder Aktoren (z.B. das Signal *Aktor.MagazinSlider*).

Die PPU-Komponente besitzt mindestens drei (Teil-)Komponenten: Dies sind ein Magazin als Eingangslager, eine Rampe als Ausgangslager und ein Kran zur Kommissionierung von Werkstücken. Zur Produktion werden Werkstücke durch das Magazin zu einer Aufnahmeposition für den Kran geschoben, was durch einen digitalen Mikroschalter wahrgenommen wird. Wenn der Kran daraufhin die Position des Magazins erreicht hat, wird das Werkstück durch den Kran mit einem Vakuumgreifer aufgenommen. Daraufhin wird es von der unteren an die

[1] http://www.dfg-spp1593.de/
[2] https://www.ais.mw.tum.de/en/research/equipment/ppu/

obere Sensorposition des Krans gefahren. Der Kran muss sich um 90° gegen den Uhrzeigersinn drehen, um Werkstücke zur Position der Rampe zu transportieren. Dort wird der Kran heruntergefahren und das Werkstück auf die Rampe gelegt. Damit ist die Produktion der Werkstücke abgeschlossen.

In einigen Experimenten werden wissenstragende Komponenten für mehrere PPU-Komponenten verwendet. Die erste und zweite PPU-Komponente produzieren zunächst Werkstücke aus Kunststoff, wohingegen die dritte PPU-Komponente ausschließlich metallische Werkstücke produziert. Diese beiden Urzustände werden als Grundstufe A1 (erste PPU-Komponente), Grundstufe B1 (zweite PPU-Komponente) und Grundstufe C1 (dritte PPU-Komponente) bezeichnet.

Während die zweite und dritte PPU-Komponente in ihren Grundstufen verbleibt, verändert sich die erste PPU-Komponente über mehrere Ausbaustufen. In Ausbaustufe A2, produzieren diese nun beide Werkstückausprägungen. Metallische Werkstücke werden dabei zu Beginn des Produktionsvorgangs im Magazin durch einen induktiven Sensor erkannt. Diese Erkennung wird auch in Grundstufe C1 der dritten PPU-Komponente verwendet.

In einer weiteren Ausbaustufe A3 wird ein Stempel hinzugefügt, der das Prägen von Werkstücken erlaubt. Der Prägevorgang wird nur für metallische Werkstücke durchgeführt. Dazu passieren die metallischen Werkstücke die Rampe am Kran hängend und werden insgesamt um 180° gegen den Uhrzeigersinn zum dahinterliegenden Stempel transportiert. Dort werden die Werkstücke abgelegt und ein Schiebezylinder des Stempels zieht das Werkstück ein. Anschließend wird das Werkstück geprägt und wieder ausgefahren. Das geprägte Werkstück wird dann durch den Kran, der am Stempel gewartet hat, aufgenommen und nach einer 90° Drehung im Uhrzeigersinn an der Rampe abgelegt.

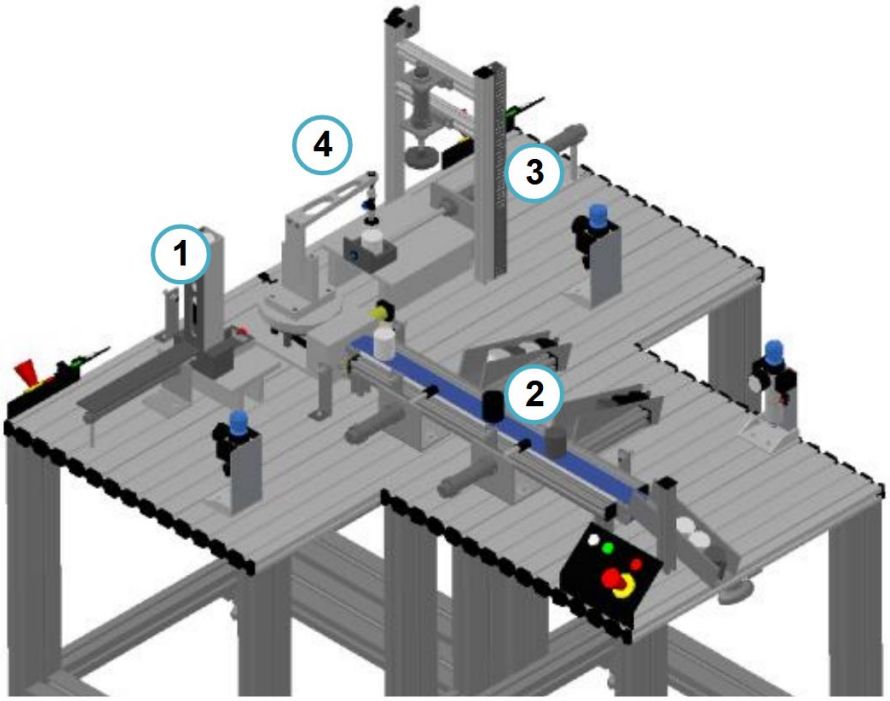


Abbildung 6.1: Schematische Darstellung der PPU-Komponente mit den Komponenten der Ausbaustufe A5: Magazin (1), Förderband (2), Stempel (3) und Kran (4)[3]

[3]Darstellung aus dem technischen Report [VHLFF] entnommen.

Mit der Ausbaustufe A4 wird eine Optimierung des Kranverhaltens durchgeführt. Dazu wird nach dem Einlegen eines metallischen Werkstücks am Stempel geprüft, ob ein Plastikwerkstück am Magazin vorhanden ist. Ist dies der Fall nutzt der Kran die Prägezeit und dreht sich im Uhrzeigersinn um das Plastikwerkstück aufzunehmen und an der Rampe abzulegen. Danach transportiert er das geprägte Werkstück vom Stempel zur Rampe. Ist während der Prägung das nächste verfügbare Werkstück am Magazin ein metallisches Werkstück wartet der Kran am Stempel bis der Prägevorgang beendet ist.

In der letzten Ausbaustufe A5 wird ein Förderband eingeführt, das die Rampe ersetzt. Das Förderband hat drei Ausgangsrampen: In eine der Rampen rutschen die Werkstücke am Ende des Förderbands. Die Befüllung der beiden weiteren Rampen erfolgt über die zwei neben dem Förderband aufgestellten Pneumatikzylinder. Diese schieben Werkstück vom Förderband in die gegenüberliegenden Rampen. Ein optischer Sensor am Anfang des Förderbands erkennt dazu das Werkstück und steuert die Pneumatikzylinder nach einer Zeitvorgabe. Abbildung 6.1 zeigt die letzte Ausbaustufe A5 in einer schematischen Darstellung. Diese enthält das Magazin (1), das Förderband (2), den Stempel (3) und den Kran (4).

Für die Diskussion der Ergebnisse werden für einige Experimenten der Evaluation noch weitere Szenarien der PPU-Komponente untersucht. Diese weiteren Szenarien sind im technischen Report der Fallstudie [VHLFF] detailliert erläutert. Darin finden sich auch weitere Details der hier verwendeten Ausbaustufen.[4] Weitere Details zur Verwendung der Laboranlage im Rahmen der Evolutionsforschung finden sich in Vogel-Heuser et al. [VHDF+14, VHFF+15].

6.2.2 Supermarktkassen mit Cloud-Verbindung als CBCPS

Als zweite Fallstudie dient die Common Component Modelling Plattform (CoCoME), welche in Herold et al. [HKW+08] vorgestellt und in Heinrich et al. [HGH+15] erweiternd beschrieben ist. Die Fallstudie wurde als Benchmark für komponentenorientierte Systeme konstruiert und für die Evolutionsforschung erweitert [HGH+15].

Das System bildet den Verkauf in einem Supermarkt ab. Das als Fallstudie für CBCPSe verwendete System besteht aus Softwarekomponenten, die mit Hardwarekomponenten einer Supermarktkasse und dem Wirtschaftsinformationssystem mit einer Lagerhaltung in einer Cloud-Infrastruktur verbunden sind. Es wird eine Filiale betrachtet, die über mehrere Kassen verfügt. An diesen scannt ein Kassierer die Waren ein und realisiert den Verkaufsprozess mit Anfragen an die Komponenten und die Cloud-Infrastruktur. Die Fallstudie fokussiert dabei insbesondere auf den cyber-physischen Teil der Hardware- und Softwareinteraktionen innerhalb der Kasse.

Für die Arbeit wurde eine Implementierung[5] des CoCoME Systems mithilfe des Pradigmas (und der entsprechenden Implementierung) der *Aktiven Komponenten* vorgenommen[6] Darin kommunizieren die Komponenten ereignisbasiert über Dienstaufrufe.

Die so realisierte Fallstudie besitzt damit sechs Komponenten, die als Systemarchitektur von *Aktiven Komponenten* in Abbildung 6.2 gezeigt sind. Vier dieser *Aktiven Komponenten* sind mit einer Hardwarekomponente verbunden.[7] Diese Komponenten sind in Abbildung 6.2 durch Hardware-Symbole gekennzeichnet. In der Fallstudie sind dies die Benutzerschnittstelle

[4] Die Grundstufen entsprechen Szenario 0 mit Werkstücken aus Kunststoff bzw. Metall; Ausbaustufe A2 entspricht Szenario 2; Ausbaustufe A3 entspricht Szenario 3; Ausbaustufe A4 entspricht Szenario 7; Ausbaustufe A5 entspricht Szenario 10

[5] Die Implementierung wurde zusammen mit Florian Abt im Rahmen seiner Tätigkeit als Hilfswissenschafter am Arbeitsbereich VSYS durchgeführt.

[6] https://github.com/cocome-community-case-study

[7] Im Rahmen der Evaluation wurden die Hardwaregeräte (softwaregestützt) simuliert.

zum Kassierer und Kunden (*KassenGUIController*), ein Scanner zum Scannen des Barcodes der Produkte (*BarcodeScanController*), ein Drucker zum Ausdrucken des Kassenbelegs (*BelegDruckController*) und eine Geldkasse (*GeldkassettenController*). Die *Kasse* ist eine Komposition dieser Komponenten. Der Ablauf des Verkaufsprozesses wird durch die *Kassenapplikation* gesteuert, die auch die Anfragen an die Cloud-Komponente des *Bestands* stellt.

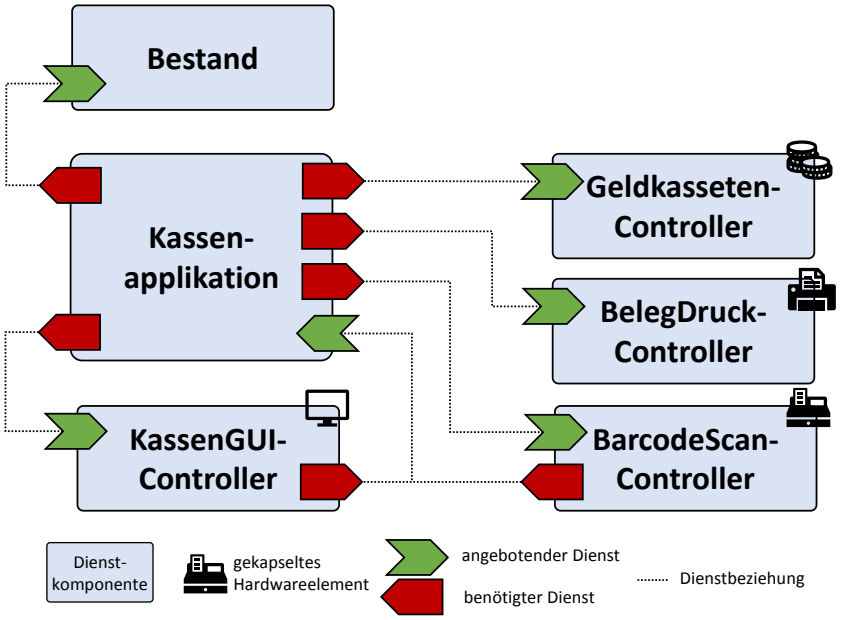


Abbildung 6.2: Aktive Komponentenarchitektur der Fallstudie für CBCPSe

Die Fallstudie hat eine Grundstufe und eine Ausbaustufe. In der Ausbaustufe besteht mit einer zusätzliche Softwarefunktionalität der Kassenapplikation eine Variante für die Möglichkeit des Scannen des Barcodes. Mit dieser Funktionalität ist es möglich, den letzten eingescannten Gegenstand erneut einzugeben, ohne dass das Produkt physisch abgescannt werden muss. Dies hat eine veränderte Aufrufreihenfolgen zur Folge.

Das Sequenzdiagramm in Abbildung 6.3 zeigt einen Verkaufsprozess der Ausbaustufe, der beide Varianten der Produkteingabe enthält. Im Verkaufsprozess werden hier drei Produkte gekauft. Die einzelnen Interaktionen im Sequenzdiagramm werden dabei immer durch eine Interaktion des Kassierers mit der Hardware angestoßen. Dies ist z.B. das Scannen mit dem Barcodescanner oder die Eingabe an der Benutzerschnittstelle. Die Hardwarekomponente besitzen eine Lebenszyklusmethode, durch die die Softwareaufrufe im CPS getätigt werden. Das Diagramm zeigt eine Sequenznummer für alle Hardwareinteraktionen und die damit verbundenen Aufrufe. Alle getätigten Aufrufe sind dabei synchron, wobei einige Aufrufe direkt zurückgegeben werden und bei anderen zunächst weitere Aufrufe stattfinden.

Ein beispielhaften Verkaufsprozess zeigt Abbildung 6.3. Dieser wird vom Kassierer durch die Benutzerschnittstelle gestartet. Der Controller der Benutzerschnittstelle startet die Kassenapplikation und dann den Belegdrucker. Nun können Produkte eingescannt werden. Diese werden durch den Barcodescanner erkannt. Immer wenn ein Produkt erkannt wird, wird die Kassenapplikation über einen entsprechenden Aufruf mit einem Ereignis informiert. Daraufhin werden die zum Barcode hinterlegten Information des Produktes über die Cloud-Infrastruktur abgefragt. Mit diesen Informationen wird der interne Warenkorb aktualisiert und der Belegdrucker über das gekaufte Produkt informiert. Im Beispielprozess wird dies für zwei verschiedene Produkte gemacht.

Für das dritte Produkt wird die zusätzliche Funktionalität der Ausbaustufe verwendet (rot markiert in Abbildung 6.3). Der Kassierer nutzt dabei eine Taste auf seiner Benutzerschnittstelle, die das zuletzt eingescannte Produkt dem Warenkorb erneut hinzugefügt. Entsprechend muss hier der Barcodescanner nicht verwendet werden und es gibt keine Abfrage der Produktdaten.

Wenn der Kassierer den Verkaufsvorgang abschließt, muss er eine Zahlungsmethode wählen. In der Fallstudie wird eine Barzahlung verwendet. In diesem Fall gibt der Kassierer die entsprechend eingezahlte Geldmenge ein und die Kassenapplikation berechnet den zurückzugebenden Geldbetrag. Beides wird neben der Kassenapplikation und dem Belegdrucker auch der Geldkassette mitgeteilt. Diese öffnet sich daraufhin. Wenn die Geldkassette manuell geschlossen wird, schließt die Kassenapplikation den Verkaufsvorgang ab. Dieser Vorgang wird auch im Bestand aktualisiert.

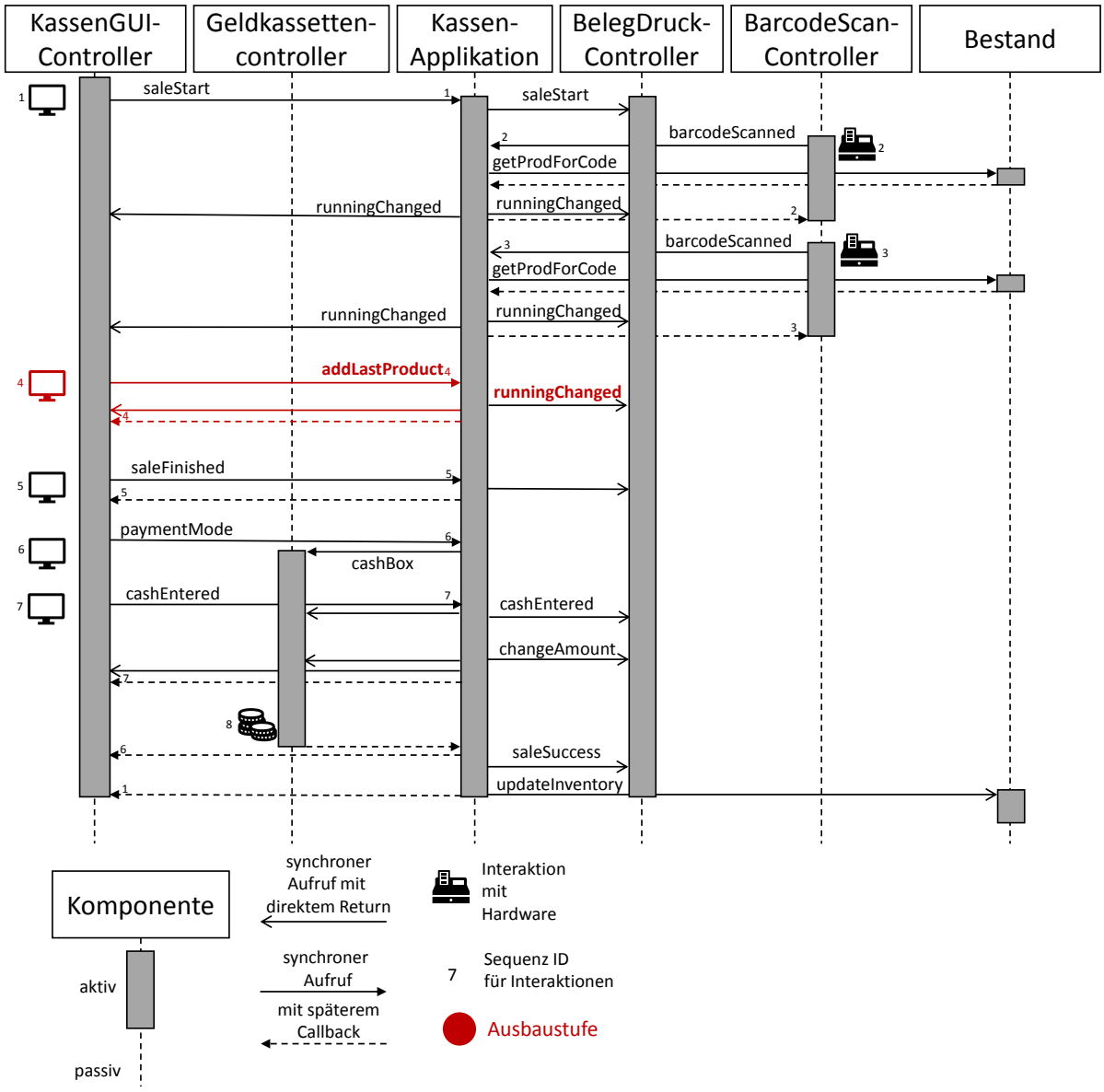


Abbildung 6.3: Beispielhafter Verkaufsprozess der Fallstudie für CBCPSe

Zusammenfassung der Fallstudien

Durch die vorgestellten Fallstudien sollen die Evaluationsziele und damit der Ansatz dieser Arbeit bewertet werden. Dafür sollten sie ein breites Spektrum an CPSen abdecken. Die Fallstudien besitzen deshalb unterschiedliche Schwerpunkte, welche zusammen die verschiedenen Kategorien von CPSen abdecken.

Die im Grundlagenkapitel gezeigten Unterscheidungsmerkmale von CPSen nach Tongren et al. [TBC+14] (siehe Abschnitt 2.4.3) werden durch die Fallstudien dabei wie folgt abgedeckt: Mit der *Pick-and-Place-Unit* wird ein sehr *hardwarenahes* CPSs betrachtet, welches stark an den Hardwareprozess gekoppelt ist. Das *Kassensystem* hat demgegenüber einen größeren Fokus auf das Softwaresysteme und besitzt klare Schnittstellen zu den Hardwaregeräten. Während die *Pick-and-Place-Unit* ein *geschlossenes CPS* mit hohem *Automatisierungsgrad* darstellt, interagiert das *Kassensystem* mit dem Kassierer, der grundsätzlich alle Interaktionen von außen anstößt. Bei der *Verteilung* besitzen beide Fallstudien eine Kombination aus zentraler Steuerung und weiteren interagierenden Systemen. Dabei nimmt die Fallstudie der *Pick-and-Place-Unit* insbesondere eine horizontale *Integration* von PPU-Komponenten vor und das *Kassensystem* mit der Cloud-Anbindung eine vertikale Integration. Die *Adaptivität* ist in beiden CPSen nicht stark ausgeprägt, wobei diese von der Evolutionsunterstützung auch nicht betrachtet wird. Denn eine gewisse Form der Adaptivität wird durch die Koevolution und die Vorschläge auf Ebene des vorgestellten Ansatzes angeboten. Durch diese gute Abdeckung der Unterscheidungsmerkmale nach Tongren et al. können die zwei Fallstudien deshalb als gute Repräsentation für CPSe gelten.

6.3 Experimentelle Evaluation

Nachfolgend werden Experimente bzw. Experimentreihen für die identifizierten Ziele der Evaluation durchgeführt. Für jedes Experiment wird dabei zunächst der *Experimentaufbau* beschrieben. Durch eine jeweils anschließende Machbarkeitsstudie (*proof-of-concept*) soll die Durchführung der Experimente demonstriert werden sowie ein besseres Verständnis der Methoden und Ergebnisse vermittelt werden. Die Machbarkeitsstudie führt dazu die wesentlichen Teile des Experiments an einem konkreten Beispiel der Fallstudie durch. Danach werden die durch das gesamte Experiment erzielten *Ergebnisse diskutiert*. Eine *Bewertung* schließt dann jedes definierte Ziel der Evaluation ab.

Die Experimente umfassen dabei nur eine begrenzte Komplexität und Anzahl an Stichproben, weshalb bei den Feststellungen grundsätzliche Beobachtungen gemacht werden können, aber in der Regel keine Signifikanz der Aussagen erreicht wird. Eine solche Signifikanz ist bei einer qualitativen Evaluation von Softwareansätzen nur schwer oder gar nicht zu erreichen. Dennoch reichen die Beobachtungen aus, um auf einer fundierten Datenbasis allgemeine Aussagen über die in dieser Arbeit vorgestellten Ansätze zu tätigen und die Machbarkeit der Ansätze aufzuzeigen.

6.3.1 Validierung der Evolutionsbeschreibung

Die wesentlichen Aspekte von Evolutionsschritten werden in dieser Arbeit durch das CRI-Modell beschrieben (siehe Abschnitt 3.2.4). Operativ wird diese Beschreibung dann auf die spezifischeren Anwendungsdomänen von CPPSen und CBCPSen angewendet (siehe Abschnitt 4.2.2). Diese Beschreibung wird nachfolgend anhand der Fallstudien evaluiert.

Beschreibung von verändernder Evolution eines CPPSs

In diesem Experiment wird untersucht, inwiefern vorhandene Szenarien von Evolution durch das vorgeschlagene CRI-Modell für Evolutionsschritte abgebildet werden können.

Experimentaufbau: Für das Experiment wurde das CRI-Modell auf die Evolutionsszenarien der PPU-Komponente angewendet. Dafür wurden alle Szenarien, was die Ausbaustufen als auch die zusätzlichen Szenerien des technischen Reports der Fallstudie (siehe [VHLFF]) umfasst, mit der Beschreibung von CPPSen kategorisiert. Diese Kategorisierung wurde anschließend bewertet. Es wurden insgesamt 23 Evolutionsszenarien ausgewertet.

Machbarkeitsstudie: Als Machbarkeitsstudie wird für Ausbaustufe A2 gezeigt, wie eine konkrete Anwendung auf das CRI-Modell aussieht:[8] Aufgrund von Veränderungen auf dem Wettbewerbsmarkt entsteht bei Ausbaustufe A2 in der Fallstudie der Bedarf, nun auch metallische Werkstücke mit der PPU-Komponente zu produzieren. Entsprechend handelt es sich bei dem Grund nach dem CRI-Modell um eine *Erweiterung* der Anforderungen des Produkts, also dem *Eingang* des CPPSs. Als Reaktion darauf ist es notwendig, dass ein neuer kapazitiver Sensor am Magazin installiert wird, um nun metallische Werkstücke unterscheiden zu können. Bei dieser Reaktion handelt es sich um eine Wirkung an der *Struktur*, die an der *Automatisierungstechnik* eine Änderung vornimmt. Das Ergebnis der Evolution ist damit, dass als Effekt die *Fähigkeiten* des *Magazins* erweitert werden.

[8] Details auf Basis des technischen Reports [VHLFF]

Diskussion der Ergebnisse

Tabelle 6.1 zeigt eine Kategorisierung der im Rahmen dieses Experimentes abgedeckten Evolutionsszenarien. Während für den Grund immer der entscheidende Trigger in der Fallstudie ausgewählt wurde, kann es aufgrund der Vielfältigkeit der Reaktionen und Ergebnisse dabei jeweils mehrfache Einordnungen geben.

Grund		**Reaktion**		**Ergebnis**	
Trigger	**23**	**Wirkung**	**37**	**Effekt**	**41**
Wartung	2	*Struktur*	13	*Zeitbezogene Größen*	8
Verbesserung	8	*Systemlogik*	14	*Produktionsvolumen*	8
Erweiterung	13	*Umgebung*	10	*Topologiegrößen*	10
				Fähigkeitsgrößen	19
Trigger von	**23**	**Wirkung an**	**52**	**Effekt auf**	**40**
Eingang	3	*Mechanik*	16	*Eingang*	3
Innerhalb	9	*Software*	19	*Magazin*	3
Ausgang	2	*Automatisie-*		*Kran*	8
Umgebung	9	*rungstechnik*	17	*Stempel*	2
				Förderband	14
				Ausgang	5
				Umgebung	5

Tabelle 6.1: Anzahl der Gründe, Reaktionen und Ergebnisse der Kategorisierung von Evolutionsszenarien

Die Tabelle zeigt, dass die Evolution hauptsächlich in Erweiterungen und Verbesserungen des CPPSs selbst oder seiner Umgebung begründet liegt. Dies resultiert aus dem Fokus der zugrundeliegenden Evolutionsszenarien, in denen weder Wartungsaktivitäten, noch größere Änderungen am Produkt selbst avisiert sind. Die Reaktionen und die Disziplinen an denen diese auftreten zeigen keine klaren Tendenzen. Dies lässt sich darauf zurückführen, dass eine Fallstudie gewählt wurde, die als Benchmark für Evolutionsansätze ausgelegt ist und dadurch eine gleichmäßige Auswahl verschiedener Szenarien der Evolution gewünscht ist. Dem Experiment nach wird diese Vorgabe erfüllt. Die Ergebnisse beim *Effekt* zeigen einen Schwerpunkt auf Fähigkeitsgrößen. Dieser begründet sich darin, dass die Fallstudie sukzessive aufgebaut und in ihrer Komplexität erhöht wird. Bezüglich der Ortsdimension zeigt die Tabelle auch die einzelnen Komponenten. Hierbei stechen insbesondere Förderband und Kran hervor, da diese bei dem zuvor genannten Aufbau die meisten Möglichkeiten einer Komplexitätserhöhung bieten. Als ein möglicher Rückschluss des Nutzers aus der Klassifizierung sollten die Erweiterbarkeit insbesondere in diesen Komponenten gewährleistet werden.

Um den Nutzen einer solchen Klassifizierung zu zeigen, wurde über die Kreuzmatrix in Tabelle 6.2 bestimmt, welche *Trigger* und *Wirkungen* jeweils zu welchen *Ergebnissen* führen.

Trigger x Von / Effekt	***Wartung*** *n = 2*	***Verbesserungen*** *n = 8*	***Erweiterung*** *n = 13*
Eingang	0 (0%)	0 (0%)	3 (23%)
Innerhalb	2 (100%)	6 (75%)	1 (8%)
Ausgang	0 (0%)	0 (0%)	2 (15%)
Umgebung	0 (0%)	2 (25%)	7 (54%)
Zeitbezogene Größen	0 (0%)	5 (63%)	2 (15%)
Produktionsvolumen	1 (50%)	4 (50%)	2 (15%)
Topologiegrößen	0 (0%)	4 (50%)	5 (38%)
Fähigkeitsgrößen	2 (100%)	5 (63%)	11 (85%)

Tabelle 6.2: Oben: Anzahl (und Prozent) der Trigger, die von einer bestimmten Ortskategorie getriggert wurden; Unten: Anzahl (und Prozent) der Trigger, die den gegebenen Effekt hervorrufen

Verbesserungen werden zumeist aufgrund der PPU-Komponente selbst durchgeführt und führen zu Änderungen in allen betrachteten nichtfunktionalen Eigenschaften. Erweiterungen sind dem entgegen zumeist durch die Umgebung der PPU-Komponente oder seiner produzierten Produkte bedingt. Entsprechend wird in 85% der Änderungen die Fähigkeit des Systems erweitert und die Änderungen schlagen sich seltener auf zeitliche oder produktvolumenbezogene Metriken nieder. Eine solche Information gibt Hinweise darauf, welche Metriken für welche evolutionären Änderungen von besonderem Interesse sind.

Eine zweite Auswertung bezüglich der Reaktionen in Tabelle 6.3 spiegelt zusätzlich die als Grundlage angenommene Interdisziplinarität von CPPSen wider. Denn die überwiegende Anzahl von Reaktionen betreffen alle beteiligten Kategorien. Dies gilt im Experiment für alle Änderungen, wobei Systemlogikanpassungen immer die Software einschließen.

Reaktion x An / Effekt	***Struktur*** *n = 13*	***Systemlogik*** *n = 14*	***Umgebung*** *n = 10*
Mechanik	11 (85%)	11 (79%)	7 (70%)
Automatisierungstechnik	10 (77%)	9 (64%)	9 (90%)
Software	11 (85%)	14 (100%)	8 (80%)

Tabelle 6.3: Anzahl (und Prozent) der Reaktionen, die Auswirkung auf eine bestimmte Disziplin haben.

Beschreibung von erhaltender Evolution eines CBCPS

Im Experiment wird untersucht, inwiefern Reports von Systemfehlern durch das vorgeschlagene CRI-Modell abgebildet werden können.

Experimentaufbau: Für die Fallstudie des Kassensystems liegen keine Evolutionsszenarien vor, weshalb die Anwendbarkeit der Beschreibung von Evolutionsschritten anhand von Fehlerreports von großen Softwarefirmen durchgeführt wurde[9]. Diese beschreiben, wie bei der Spezifizierung des CRI-Modells für CBCPSe vorgesehen, das Ziel einer erhaltenden Evolution. Das Experiment umfasst 51 Reports, die kritische Systemausfälle zusammenfassen und ihre Auswirkungen darstellen. Eine detaillierte Darstellung der durchgeführten Methodik und der Reports findet sich in Haubeck et al. [HPR+18, RPH+17].

Machbarkeitsstudie: Als Machbarkeitsstudie wird für einen ausgewählten Report die entsprechende Kategorisierung erläutert: Dem Report ist u.a. zu entnehmen, dass es aufgrund eines Fehlers in den Speicherressourcen eines Drittanbieters zu einem Systemausfall kam. Denn durch diesen haben sich Speicherressourcen bei hoher Auslastung immer wieder mit Verarbeitungskomponenten neu verbunden. Nach der Änderung am Netzwerk trat dieser Fehler bei mehreren Verarbeitungskomponenten gleichzeitig auf, wodurch das Netzwerk überlastetet wurde. Entsprechend war der Trigger eine Veränderung der *Hardware innerhalb* des CBCPSs. Dieser Trigger verursachte als Reaktion eine negative *Wirkung* der Kommunikation, die an den *Kommunikationskanälen* des CBCPSs zu einem Ausfall führte. Als Effekt musste das CBCPS durch die entstandene Überlast hochskalliert werden, was u.a. zu erhöhten *Kosten innerhalb* des CBCPSs führte.

Diskussion der Ergebnisse

Tabelle 6.4 zeigt zusammenfassend die Ergebnisse der Kategorisierung. Dabei sind jeweils mehrere Trigger, Wirkungen und Effekte möglich, wobei hier nur starke, beschriebene Einflüsse gewertet wurden.

Grund		**Reaktion**		**Ergebnis**	
Trigger	**65**	**Wirkung**	**65**	**Effekt**	**65**
Hardware	22	*Kommunikation*	24	*Datenintegrität*	37
Software	21	*Systemlogik*	23	*Zeitliche Kosten*	27
Nutzungsmuster	22	*Ressourcen*	18	*Monetäre Kosten*	1
Trigger von	**65**	**Reaktion an**	**65**	**Effekt auf**	**65**
Upstream	22	*Komponente*	4	*Downstream*	0
Innerhalb	36	*Kanal*	43	*Innerhalb*	11
Downstream	7	*Ressource*	18	*Upstream*	54

Tabelle 6.4: Anzahl der Gründe, Reaktionen und Ergebnisse der Kategorisierung von Reports

Wie erwartet werden Veränderungen in CBCPSen hauptsächlich durch das CBCPS selbst oder ihren Upstream verursacht. Eine Reaktion findet dabei zumeist auf den Komponenten statt, da diese die komplexeste Logik enthalten. Deshalb ist als Schlussfolgerung eine Evolutionsunterstützung über Modelle und Metriken insbesondere dieser Komponenten erforderlich.

[9] Die Analyse wurde im Rahmen einer Masterarbeit mit Kim Reichert durchgeführt.

Die Kanäle der CBCPSe scheinen den Reports zufolge selten zu kritischen Systemausfällen zu führen. Bezüglich der Ergebnisse werden in den Reports besonders Upstream-Effekte auf Zeit und Kosten genannt. Dieses ist allerdings vermutlich auf die Beschreibung der Veränderungen in den Reports zurückzuführen, wobei monetäre Kosten oder Einflüsse auf den Downstream nicht kommuniziert werden.

Der Nutzen der Kategorisierung wird im Rahmen der Erkennung von Abweichungen demonstriert. Dafür wird der Zusammenhang zwischen Trigger und Wirkung in Tabelle 6.5 untersucht. Die Auswertung zeigt, dass Abweichungen der Kommunikation grundsätzlich durch eine Veränderung der Hardware getriggert werden. Ressourcenengpässe treten hingegeben zumeist aufgrund eines veränderten Nutzungsmusters auf und basieren seltener auf Hardware- oder Softwareveränderungen. Da Nutzungsmuster bisher zumeist durch Fehler-Logs oder Anwendungsmetriken bestimmt werden, sollten den Ergebnissen des Experiments zu Folge diese verstärkt durch Metriken der Ressourcenauslastung erweitert werden (vergleiche Haubeck et al. [HPR+18]).

Trigger x Wirkung	***Software***	***Hardware***	***Nutzungsmuster***
Kommunikation	3	38	6
Systemlogik	14	4	5
Hardware	4	1	26

Tabelle 6.5: Anzahl der Wirkungen, die durch einen Trigger hervorgerufen wurden

Zusammenfassende Bewertung des ersten Evaluationsziels

Allgemein lassen sich folgende Schlussfolgerungen aus den zwei Experimenten zur Beschreibung ziehen:

- Die Anzahl an Fällen lässt keine signifikanten Aussagen zu. Dennoch hat das Experiment gezeigt, dass die Evolutionsszenarien und Reports durch das CRI-Modell abgebildet werden können. Über Kreuzmatrizen wurde demonstriert, wie aus dieser Beschreibung hilfreiche Schlussfolgerungen gezogen werden können.
- Durch die Kategorisierung kann der Evolutionsprozess analysiert werden. Dies ermöglicht es u.a. zu bewerten, welche Art von Evolutionsschritten in einem CPS auftreten. Entsprechend kann diese Beschreibung für Kontextinformationen von Evolutionsschritten verwendet werden, damit semantische Aussagen über den Grund, die Wirkung und den Effekt einer Evolution schematisch beschrieben werden können. Damit kann die Beschreibung insbesondere auch für eine post-mortem Analyse verwendet werden.
- Das Experiment hat damit gezeigt, dass die Dimensionen der Beschreibung sowohl für unterschiedliche Ziele als auch Anwendungsgebiete einer Evolution stabil bleiben und mit den Kategorien auf diese zugeschnitten werden können.

6.3.2 Einbindung von Ereignis- und Kontextquellen

Unter diesem Ziel der Evaluation wird die Einbindung von heterogenen Quellen über die entworfene Systemarchitektur evaluiert. Dabei werden zunächst die für die Evolutionsunterstützung wesentlichen Bussysteme und Dienstaufrufe untersucht. Zwei weitere kleinere Experimente zeigen, dass auch Entwicklungsartefakte, wie u.a. Softwarecode und Testartefakte, als Quellen in der wissenstragenden Komponente verwendet werden können.

Konsistente und einheitliche Zustände eines Bussystems

Dieses Experiment untersucht die Zustände, die aus einem Bussystem extrahiert werden.

Experimentaufbau: Der Experimentaufbau für die PPU-Fallstudie umfasst die (Automatisierungs)hardware und eine SPS, die über eine *EtherCAT* Verbindung angebunden ist. Für die Experimente wurde eine SoftSPS-System[10] verwendet. Diese besitzt einen OPC-Server, der die Daten über ein Gateway des *Open Platform Communication (OPC)*-Protokolls bereitstellt.

Die wissenstragende Komponente wird auf einem eigenen Rechensystem betrieben.[11] Sie verwendet eine Adapterkomponente, die einen OPC-Client über das Prosys[12]-OPC-Framework einbindet. Die aufzunehmenden Ereignisse sind in der Konfiguration der Adapterkomponente festgelegt. Das Experiment beschränkt sich auf die Aufnahme von allen binären Ereignissen der Sensoren und Aktoren. Es wird dabei eine OPC-Zykluszeit von 60 ms verwendet, weshalb alle auftretenden Ereignisse in diesem Zyklus die gleichen Zeitstempel besitzen. Die Kontextinformationen der Ereignisse sind durch eine zusätzliche Informationsquelle hinterlegt, auf der in der Adapterkomponente zugegriffen wird.

Für das Experiment besitzt jede Teilkomponente (Kran, Magazin, Stempel und Laufband), sowie die gesamte PPU-Komponente jeweils eine Repräsentationskomponente. Diese haben sich den Kontextinformationen entsprechend an der Adapterkomponente registriert und erzeugen die zu untersuchenden Zustände.

Machbarkeitsstudie: Die Machbarkeit wird am Beispiel der Grundstufe A1 des Magazins gezeigt. Dabei werden die in Tabelle 6.6 angegebenen Ereignistypen des Magazins verwendet.[13] Für diese sind die Namen der Sensoren und Aktoren mit ihren Typ- und Ortskontexten angegeben. Diese entsprechen einer endlichen Liste von Ereignistypen der Fallstudie (vergleiche Ladiges [Lad18] und die später folgende Abbildung 3.14).

Tabelle 6.6: Orts- und Zeitkontext der Ereignisse des Magazins in Grundstufe A1

Ereignis	**Ortskontext**	**Typkontext**
S.SliderNotMovedOut	Sensor-XPosition	Magazin
S.SliderMovedOut	Sensor-XPosition	Magazin
A.MagazinSlider	Aktor-Bewegung	Magazin
S.WorkpieceReady	Sensor-Werkstück-Erkennung	Magazin
S.CranOnMagazin	Sensor-XPosition	Magazin

[10] CODESYS: https://de.codesys.com
[11] Notebook, 2 Kern CPU, 8 GB RAM
[12] https://www.prosysopc.com/
[13] Manuelle Steuerungsereignisse wurden vernachlässigt.

Auf dem Bussystem wurden nach der Initialisierung, die im linken Abschnitt von Tabelle 6.7 dargestellten Ereignisse beobachtet. Die Adapterkomponente bildet die Funktionalitäten eines Informationsartefakts ab indem es diese OPC-Ereignisse um die Kontexte der statischen Kontextquelle anreichert. Die Repräsentationskomponenten erzeugen Zustandstupel entsprechend der vorgestellten Methodik. Ein Wertetypobjekt ist beispielsweise zum Zeitpunkt 5621 Millisekunden das Zustandstupel $(1, 0, 1, 0, 0)$ zusammen mit dem beobachten Ereignistupel $(A.MagazinSlider,\ 1,\ 5621ms,\ Magazin,\ Aktor - Bewegung)$.

Tabelle 6.7: Ausschnitt der Ereignisse und Zustandstupel in der Grundstufe A1

			S.SliderNotMovedOut	**S.SliderMovedOut**	**A.MagazinSlider**	**S.WorkpieceReady**	**S.CraneOnMagazin**
Zeit	**Signalname**	**Wert**	**Zustandstupel**				
0	Anfangszustand	-	1	0	0	0	0
5621	A.MagazinSlider	1	1	0	1	0	0
5742	S.SliderNotMovedOut	0	0	0	1	0	0
6399	S.SliderMovedOut	1	-	-	-	-	-
6399	S.WorkpieceReady	1	0	1	1	1	0
8620	S.CranOnMagazin	1	0	1	1	1	1
9640	S.SliderMovedOut	0	-	-	-	-	-
9640	S.WorkpieceReady	0	0	0	1	0	1
9699	A.MagazinSlider	0	0	0	0	0	1
9821	S.SliderNotMovedOut	1	1	0	0	0	1
11921	S.CranOnMagazin	1	1	0	0	0	0

Diskussion der Ergebnisse

Für das Experiment wurden jeweils drei unterschiedliche Werkstoffkonstellationen mit jeweils sechs Werkstücken untersucht. Da eine Koevolution mit Modellen erfolgen soll, ist neben der Aufnahme von konsistenten Zuständen auch die Einheitlichkeit der Zustandsabfolgen von besonderer Bedeutung, da diese zu Änderungen auf der Modellebene führen. Tabelle 6.8 zeigt für das Experiment die Anzahl der Zustände und die Einheitlichkeit der Reihenfolge. Die Anzahl der Zustände gibt an, wie viele unterschiedliche Zustände erzeugt wurden. Einheitlichkeit bedeutet hier, wie viele der Zustände (in Prozent) eine vollständig gleiche Reihenfolge hatten.

Die Anzahl der Zustände steigt erwartungsgemäß mit deren Ausbaustufe. Höhere OPC-Zykluszeiten besitzen weniger Zustände, da Ereignisse öfter zusammengeführt werden. Genauso wird die Einheitlichkeit davon beeinflusst. Denn bei hoher OPC-Zykluszeit werden zeitlich differierende Verhalten in gleiche Zustände einsortiert, da diese im selben Beobachtungszyklus passieren. Verschiedene Ereignisreihenfolgen treten insbesondere in Ausbaustufe A5 auf, da besonders das Abstellen der Werkstücke auf dem Laufband zeitlich variiert.

Ausbaustufe	Zykluszeit	Anzahl der Zustände	Einheitlichkeit
A1	30 ms	160	98%
	60 ms	153	100%
A2	30 ms	163	97%
	60 ms	154	100%
A3	30 ms	241	86%
	60 ms	232	98%
A4	30 ms	260	88%
	60 ms	243	97%
A5	30 ms	511	69%
	60 ms	460	88%

Tabelle 6.8: Konsistenz und Einheitlichkeit der beobachteten Zustände

Dies lässt folgende Schlussfolgerungen aus dem Experiment zu:

- Die wissenstragende Komponente konnte konsistente Zustände aus den Ereignissen eines Bussystems generieren.
- Die statische Einbindung von Kontextinformationen über ein zusätzliches Kontextartefakt war problemlos möglich und erfordert einen einmaligen und geringen Aufwand. Sie stellt allerdings eine Einschränkung bei der Evolutionsunterstützung dar, da für hinzugefügte Sensoren und Aktoren immer auch der Kontext modelliert werden muss.
- Die Genauigkeit hängt weiterhin von der gewählten Technologie zur Beobachtung ab. Im gezeigten Experiment entsteht eine technisch bedingte Streuung bzw. Unschärfe der Zeitstempel. Dies ist nach einer Analyse der Experimentergebnisse auf die Latenz und Auslastung von Bussystem und Ethernetverbindung sowie insbesondere die Divergenz zwischen Zykluszeit des OPC-Servers und der SPS zurückzuführen.
- Entsprechend sind die Zustände bezüglich der Zeitstempel konsistent, aber die Zustandsfolge ist nicht immer einheitlich. 60 ms hat sich als angemessener Wert der OPC-Zykluszeit für die Fallstudie gezeigt. Dadurch ist sowohl eine grundsätzlich einheitliche Folge von Zuständen als auch eine akzeptable Abbildung des Verhaltens zu erwarten. Die wissenstragende Komponente kann dadurch in der Regel noch *weiche Echtzeitanforderungen* erfüllen.

Konsistente Nachrichtensequenzen aus Methodenaufrufen

In diesem Experiment wird anhand des Supermarktsystems gezeigt wie Dienstaufrufe in die Evolutionsunterstützung integriert werden können.

Experimentaufbau: Das Kassensystem kann aufgeteilt in zwei ausführbaren Java-Anwendungen auf einem Server bereitgestellt werden. Der erste Teil stellt hier eine Kasse dar und wurde auf einer virtuellen Maschine[14] betrieben. Um die Skalierbarkeit der Systemarchitektur zu untersuchen, wurde an dieser Stelle die Anzahl der Kassen und auch die Anzahl der Verkaufsprozesse sukzessive erhöht.

[14] 2 Kern CPU, 8 GB RAM

Daneben wurde auch die Bestandskomponente auf einer weiteren virtuellen Maschine[15] bereitgestellt. Alle Komponenten sind über Jadex-Plattformen miteinander verbunden.

Die Fallstudie ist durch die Jadex-Plattform für die Beobachtung bereits instrumentalisiert, wodurch der Start und das Ende eines jeden Methodenaufrufes beobachtet werden kann. Dabei werden die als Hardwareelemente ausgelegten Komponenten jeweils durch ein Softwaredummy ersetzt, der den Vorgang simuliert durchführen kann oder die Benutzerschnittstelle des zugrundeliegenden CoCoME Systems verwendet.

Die Ereignisse werden durch eine Adapterkomponente einer wissenstragenden Komponente verarbeitet, die auf einer zusätzlichen virtuellen Maschine[16] bereitgestellt wurde. Da eine Differenzierung nach Komponenten nicht notwendig ist, wurde nur eine Repräsentationskomponenten verwendet, die das gesamte CBCPS abbildet.

Machbarkeitsstudie: Zur Erläuterung wird das Scannen und Verarbeiten eines Produktes nach dem gegebenen Beispielprozess der Fallstudie betrachtet. Die über die Adapterkomponente beobachteten Ereignisse sind in Tabelle 6.9 gezeigt.

Tabelle 6.9: Beobachtete Ereignisse des ersten Scanvorgangs

Komponente.Aufruf	**SequenzID**	**Startzeit**	**Endzeit**
KassenApplikation.BarcodeScanned	2	812	1020
Bestand.getProductForCode	2	823	872
BelegDruckController.runningChanged	2	898	920
KassenGUIController.runningChanged	2	938	963

Über die Repräsentationskomponenten werden der Evolutionsunterstützungsmethode folgend die Nachrichtensequenz gebildet. Dazu werden die beobachteten Methodenaufrufe nach ihrer Zeit geordnet (vergleiche [Roh15]). Die darin enthaltenen Aufrufe sind in Tabelle 6.10 gezeigt. Die Aufrufe sind entsprechend an die Methoden gebunden und haben einen definierten (Aufruf-)Zeitpunkt.

Tabelle 6.10: Nachrichtensequenz des Scannen eines Barcodes

	Aufrufender	**Aufgerufene**	**Aktion**	**Zeit**
1.	$	KA.BarcodeScanned	Aufruf	812
2.	KA.BarcodeScanned	B.getProductForCode	Aufruf	823
3.	B.getProductForCode	KA.BarcodeScanned	Rückgabe	872
4.	KA.BarcodeScanned	KG.runningChanged	Aufruf	898
5.	KG.runningChanged	KA.BarcodeScanned	Rückgabe	920
6.	KA.BarcodeScanned	BS.runningChanged	Aufruf	938
7.	BS.runningChanged	KA.BarcodeScanned	Rückgabe	963
8.	KA.BarcodeScanned	$	Rückgabe	1020

[15] 4 Kern CPU, 16 GB RAM
[16] 4 Kern CPU, 16 GB RAM

Diskussion der Ergebnisse

Die Systemarchitektur entspricht an dieser Stelle einer etablierten *Black-Box*-Beobachtung, die in ihrer Funktionsweise durch das Experiment bestätigt wurde. Mit dem Experiment wurde darüber hinaus die Skalierbarkeit untersucht. Dazu wurde geprüft, ob das prototypische System bei gängiger Rechenleistung einem typischen Kassensystem standhalten kann.[17]

Über ein Simulationskomponente wurde dafür jeweils ein normalverteilter Verkaufsvorgang mit durchschnittlich 10 Produkten, die in etwa 45 Sekunden verkauft werden, mehrfach simuliert. Dies entspricht etwa dem durchschnittlichen Verkaufsprozess in deutschen Supermärkten (vergleiche [Mih14]).

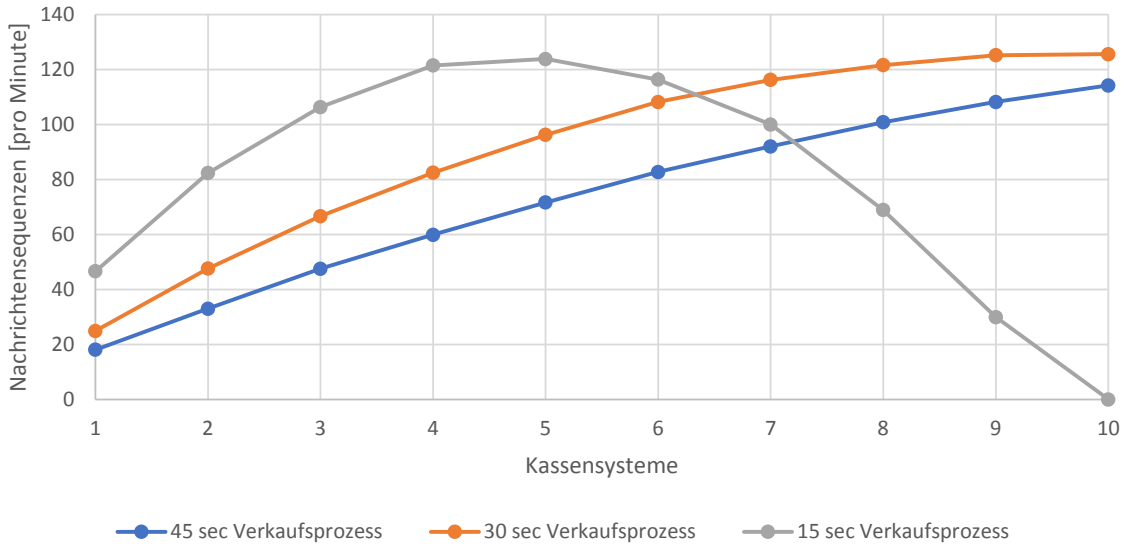


Abbildung 6.4: Nachrichtensequenzen pro Sekunde bei unterschiedlicher Anzahl von Kassensystemen und Dauer des Verkaufsprozesses

Wie Abbildung 6.4 mit der blauen Lastkurve zeigt, steigt die Anzahl der Nachrichtensequenzen, die die wissenstragende Komponente generiert, stetig an und es ist bei 10 Kassensystemen keine maximale Grenze ersichtlich. Dieses Maximum wird allerdings bei gleicher Produktanzahl und reduzierter Zeit von 30 Sekunden pro Verkaufsprozess erreicht (orange Lastkurve in Abbildung 6.4).[18] Die maximale Leistung wird bei einer weiten Reduktion auf 15 Sekunden pro Verkaufsprozess im Experiment noch deutlicher. In der Simulation zeigt sich auch der für den Durchsatz eines Softwaresystems typische Einbruch, der in der wissenstragenden Komponente aus der Überflutung der Adapter- und Repräsentationskomponenten resultiert. Entsprechend erhöht sich auch die Antwortzeit bei Aufrufen der wissenstragenden Komponente.

Dies lässt folgende Schlussfolgerungen aus dem Experiment zu:

- Nachrichtensequenzen können über die Systemarchitektur in die Evolutionsunterstützung eingebunden werden.
- Die wissenstragende Komponente kann für das Supermarktsystem ein realistisches Szenario performant abbilden.

[17] Die Simulation wurde in Zusammenarbeit mit Florian Abt im Rahmen seiner Tätigkeit als Hilfswissenschaftler an der Universität Hamburg durchgeführt.

[18] Hierfür wurde die Zeit, die der simulierte Verkäufer für das Scannen und Kassieren benötigt, verringert. In der Simulation wird angenommen, dass immer Kunden verfügbar sind.

- Es zeigen sich für die Skalierung eine typische Lastkurve und die daraus resultieren Antwortzeiten. Im Experiment liegt dabei die maximale Anzahl von Nachrichten bei etwa 125 pro Minute.
- Der Durchsatz hängt maßgeblich von der Anzahl der eingehenden Ereignisse und der Komplexität der Zustandsverarbeitung ab. Für komplexere Systeme oder Modelle wäre deshalb eine Optimierung des implementierten Prototyps notwendig.

Analyse des Softwarecodes zur Kontextbestimmung

Mit diesem ergänzenden Experiment wird untersucht, ob Kontexte aus dem Softwarecode einer Anwendung von *Aktiven Komponenten* durch das vorgeschlagene Verfahren extrahiert werden können.

Experimentaufbau: Für die Analyse des Softwarecodes wurde zunächst der Quellcode der Implementierung der CBCPS-Fallstudie in eine Eclipse-Programmierumgebung[19] importiert. Um den abstrakte Syntaxbaum zu traversieren, wurde das Projekt Java-Development-Tools[20] verwendet. Dieses Projekt bietet eine Klasse *ASTVisitor* an, die abstrakte Syntaxbäume über das Entwurfsmuster des Besuchers (siehe [GJHV11]) traversieren kann. Die abstrakte Klasse wurde um die Regeln für Systeme aus *Aktiven Komponenten* (siehe Abschnitt 3.3.2) erweitert, um Kontextinformationen für die Fallstudie zu extrahieren.

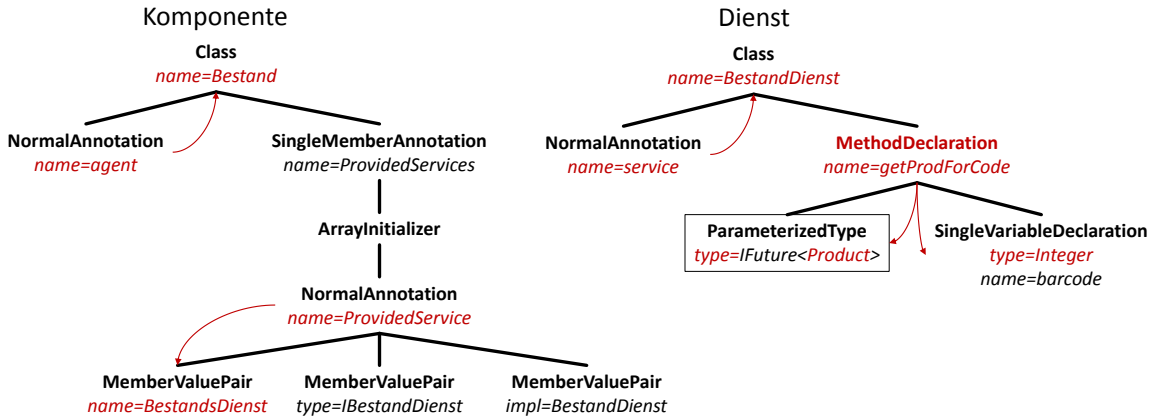


Abbildung 6.5: Relevante Teile der abstrakten Syntaxbäume der Komponente *Bestand* und seines Dienstes

Machbarkeitsstudie: Für den beispielhaften Verkaufsprozess ergibt sich damit der in Abbildung 6.5 gezeigte Ausschnitt aus dem abstrakten Syntaxbaum für die Komponente *Bestand* und ihren *BestandsDienst*. Als wesentliche Informationen zeigt der Syntaxbaum die @Agent-Annotation und die @Service-Annotation. Über die Regeln der Methodik können somit die Namen des Dienstes und der Komponenten bestimmt werden. Dies ist in der Abbildung 6.5 über Pfeile gekennzeichnet.

[19] https://www.eclipse.org/
[20] https://www.eclipse.org/jdt/

Zusammen mit den @ProvideServices{@ProvidedService(...),...}- und @RequiredServices{@RequiredService(...),...}-Annotation wird der Ortskontext extrahiert. In der Abbildung zeigt sich, dass die Komponente *Bestand* genau eine Annotation mit einem angebotenen Dienst besitzt. Diese hat den Namen *BestandsDienst* und keinen benötigten Dienst. Die Methode *getProductForCode* wird durch diesen Dienst angeboten. Somit ist der Ortskontext dieser Methode durch die Wertepaare $(komponente = Bestand)$, $(dienst = BestandsDienst)$ und $(angeboteneDienste = \{BestandsDienst\})$ gegeben.

Über den abstrakten Syntaxbaum des Dienstes lassen sich weiterhin die angebotenen Methoden beschreiben indem das *MethodDeclaration*-Element traversiert wird. Dabei lassen sich der Name, der Rückgabetyp und die Parameter über die entsprechenden Regeln identifizieren. Die Wertepaare des Typkontextes sind somit $(name = getProductForCode)$, $(rückgabe = Product)$ und $(parameter = \{Integer\})$

Diskussion der Ergebnisse

Die durch die Methode erkannten Kontextinformationen zeigt Tabelle 6.11. Durch eine manuelle Validierung ist hier festzustellen, dass alle 6 Komponenten und 6 Dienste, sowie die 15 bzw. 16 Methoden erkannt wurden. Es ist weiterhin zu erkennen, dass die Evolution von der Grundstufe zur Ausbaustufe nur eine minimale Veränderung am System vornimmt indem eine Methode hinzugefügt wird.

Kontextinformationen	**Grundstufe**	**Ausbaustufe**
Komponenten	6	6
Dienste	6	6
Angebotenen Dienstverbindung	6	6
Benötigte Dienstverbindung	7	7
Methoden	15	16

Tabelle 6.11: Erkannte Kontextinformationen der Fallstudie für CBCPSe

Folgende Schlussfolgerungen konnte damit mithilfe dieses Experimentes gezogen werden:

- Die vorgeschlagene Methode ist geeignet für Anwendungen auf Basis von *Aktiven Komponenten* wesentliche Orts- und Typkontexte zu identifizieren. Die Methode ist dabei auf *Aktive Komponenten* beschränkt. Die methodische Einbindung eines solchen Verfahrens ist allerdings auf andere generische Verfahren übertragbar (vergleiche die Übersicht von Krogmann et al. [Kro12])
- Die operative Einbindung in die wissenstragende Komponente wurde nicht untersucht, wäre allerdings prinzipiell durch einen entsprechenden Adapter mit Zugang zum Quellcode möglich.
- Die Methode ist nur bei der Erstbetrachtung des Systems und nach einer vorab durchgeführten Evolution hilfreich, da dazu der Softwarecode vorliegen muss. Entsprechend hilft die Methode für Vorschläge zur Evolutionsunterstützung nur indirekt weiter indem eine gute Informationsbasis für das ursprüngliche Modell erstellt werden kann. Zukünftige Kontexte nach der Evolution sind nicht ableitbar.

Testartefakte als Quellartefakt vor der Evolution

Im zweiten ergänzenden Experiment wird die Eindung von Testfällen untersucht. Dies soll zeigen, dass auch Entwicklungsartefakte anstelle des operativen Systems als Ereignisquelle verwendet werden können.

Experimentaufbau: Im Experiment werden Informationen von sequentiellen Funktionsdiagrammen in die wissenstragende Komponente integriert.[21] Das Framework für das modellbasierte Testen nach Lochau et al [LBL+14] (siehe Abschnitt 3.3.2) wurde für sequentielle Funktionsdiagrammen konfiguriert und in einer Adapterkomponente verwendet. Die sequentiellen Funktionsdiagramme wurden für die PPU-Komponente nach der Beschreibung von Vogel-Heuser et al. [VHLFF] erstellt. Für das Experiment wird ein Funktionsdiagramm des Krans in Ausbaustufe A4 betrachtet.

Die SPS wird ohne die eigentliche Laboranlage betrieben indem das modellbasierte Testverfahren die Eingabe- und Ausgabewerte triggert. Die Konfiguration der wissenstragenden Komponente ist ansonsten analog zum Experiment für Bussysteme. Aus den generierten Zuständen der wissenstragenden Komponente werden Maschinenzustandsmodelle generiert.[22]

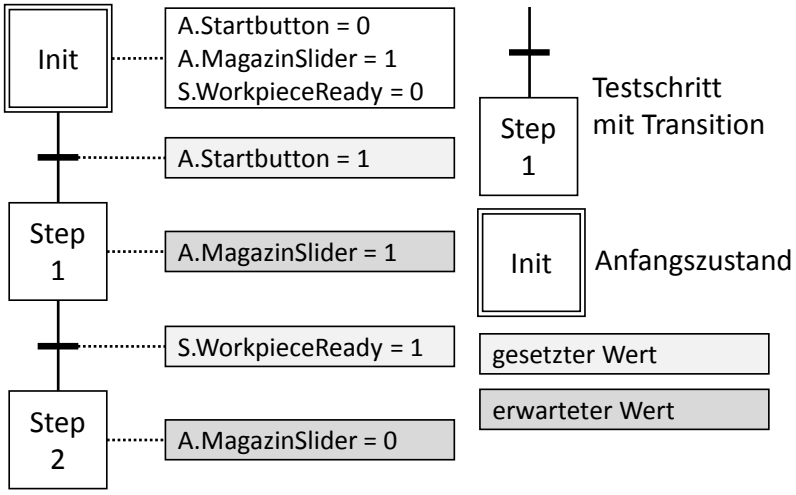


Abbildung 6.6: Testmodell mit einem Testfall für das Magazin

Machbarkeitsstudie: Abbildung 6.6 zeigt ein einfaches Testmodel. Dieses hat nur einen Testfall, weshalb das vollständige Abdeckkriterium durch den Testfall direkt erreicht wird.[23] Der gezeigte Ausschnitt des Testfalls beschreibt den Anfangszustand der Fallstudie. Er spezifiziert, dass nach der Erzeugung eines Ereignisses *A.Startbutton* die Steuerungssoftware ein Ereignis für den Aktor *A.MagazinSlider* setzten muss. Dies entspricht dem erwarteten Verhalten der PPU-Komponente, welche nach dem Start ein erstes Werkstück ins System einleitet. Wenn der Aktor gesetzt wird und dies vom Testfall erkannt wurde, wird simuliert, dass das Werkstück am Magazin nun bereit liegt. Dies erfolgt durch ein weiteres Ereignis, woraufhin erwartet wird, dass das Magazin wieder in die Ausgangssituation zurückkehrt.

[21] Eine erste Version des Experiments wurde im Rahmen der Forschungstätigkeiten zur Arbeit in Kooperation mit Jan Ladiges und Sasha Litty durchgeführt.
[22] Die Generierung wird im nächsten Evaluationsziel ausführlicher betrachtet.
[23] Das vollständige Testmodell des Experiments verwenden die Testfallgenerierung nach Lochau et al. [LBL+14], wobei als Abdeckungskriterium alle Transitionen durchlaufen werden müssen.

Die Adapterkomponente konnte durch die Beobachtung dieses Testfalls die in Tabelle 6.12 gezeigten Ereignisse aufzeichnen. Einige der gezeigten Ereignisse wurden durch die SPS ausgelöst und andere durch den Testfall selbst. Alle Ereignisse zusammen spiegeln das simulierte, interdisziplinäre Verhalten des CPPSs wider.

Tabelle 6.12: Ereignisse bei der Ausführung des Testfalles

Testschritt	**Signalname**	**Wert**
0	Anfangszustand	-
1	A.StartButton	1
1	A.MagazinSlider	1
2	S.WorkpieceReady	0
2	A.MagazinSlider	0

Diskussion der Ergebnisse

Die Experimentergebnisse zeigt Tabelle 6.13. Das im Experiment verwendete sequentielle Funktionsdiagrammen bestand aus 97 Schritten und 109 Transitionen. Durch das Abdeckungskriterium werden 109 Testfälle erzeugt. Aufgrund von Redundanzen wurden 12 dieser Testfälle zur Durchführung ausgewählt und simuliert. Die wissenstragende Komponente nimmt dabei 11 für das Verhalten des Krans relevante Ereignistypen auf.

Zur Untersuchung des Nutzens von Testfällen wurde das extrahierte Verhalten aus den Testfällen mit dem extrahierten Verhalten des operativen Systems verglichen. Es zeigt sich dabei, dass das operative System ca. 57% mehr Modellelemente generiert. Denn das für diesen Zweck generierte Maschinenzustandsmodell wächst von 23 auf 37 Stellen und von 28 auf 43 Transitionen.

Tabelle 6.13: Experimentergebnisse von Modellen aus Testfällen

Typ	**Indikator**	**Typ**	**Ergebnis**
Testfälle	Anzahl	Schritte (seq. FD)	97
		Transitionen (seq. FD)	109
		Generierte Testfälle	109
		Ausgewählte Testfälle	12
Modell (vorher)	Anzahl	betrachtete Ereignisse	11
		generierte Stellen	23
		generierte Transitionen	48
Modell (nachher)	Anzahl	Stellen	13
		Transitionen	22
Abweichung	Kategorien	Einfügung	8
		Übergreifend	3

Entsprechend lässt das Experiment folgende Schlussfolgerungen zu:

- Testfälle können, wie die Beobachtung des operativen CPSe, als Quellartefakt genutzt werden. Im Vergleich zu einem industriell eingesetzten System ist die Anzahl der Ereignisse in der Fallstudie allerdings gering.
- Nach den Ergebnissen des Experiments konnte mit ca. 64% gut die Mehrzahl der Modellelemente des Verhaltens vor der eigentlichen Inbetriebnahme durch die Testfälle generiert werden. Dies entspricht einem relevanten Einblick in das Verhalten, kann allerdings die operative Beobachtung nicht ersetzen.

Zusammenfassende Bewertung des zweiten Evaluationsziels

Durch die vier Experimente wurde gezeigt, dass unterschiedliche Quellartefakte für die Evolutionsunterstützung in die wissenstragende Komponente integriert werden können. Für ein Bussystem konnten unter einer geeigneten Konfiguration für die Fallstudie erfolgreich konsistente und einheitliche Zustandsfolgen erzeugt werden. Genauso konnten mit der Beobachtung von Dienstaufrufen Nachrichtensequenzen als Grundlage einer Evolutionsunterstützung gewonnen werden. Das Experiment ließ für die Fallstudien eine ausreichende Skalierbarkeit erkennen. Dabei konnte demonstriert werden, dass beide Verfahren geeignete Zustände für eine weitergehende Koevolution und Evolutionsunterstützung für Laufzeitmodelle bereitstellen können.

Durch die Simulation von Testfällen und die Analyse des Softwarecodes von CPPSen wurde zudem gezeigt, dass auch vorhandene Artefakte der Entwicklung in die Evolutionsunterstützung integriert werden können. Genauso konnte eine Kontextquelle über den Quellcode geschaffen werden, der Kontexte von Methoden aus dem Quellcode extrahiert. Damit ist es möglich auch neben dem operativen System Ereignisse in die Methodik dieser Arbeit einzubinden. Entsprechend konnte das Ziel der Integration von unterschiedlichen Artefakten in die Evolutionsunterstützung erfolgreich evaluiert werden.

6.3.3 Koevolution von Modellartefakten

Nachfolgend wird die konzipierte Koevolution jeweils für CPPS- und CBCPS-Modelle evaluiert.

Spezifische Modellartefakte von CPPSen

Zunächst wird die Koevolution bezüglich der Einbindung von Materialfluss- und Maschinenzustandsmodellen untersucht.

Experimentaufbau: Um spezifische Modelle von CPPSen einzubinden, wird der Aufbau für Bussysteme verwendet. Neben der Adapterkomponente und den Repräsentationskomponenten besitzt die wissenstragende Komponente zusätzlich fünf Wissenskomponenten für CPPS-Modelle. Dies sind ein Materialflussmodell des gesamten CPPS und jeweils ein Maschinenzustandsmodell von Magazin, Kran, Stempel und Laufband. Die Konfiguration der Modelle wurde mit entsprechenden Filterregeln entlang des Ortskontextes der Ereignisse vorgenommen.

Die Ausbaustufen der Fallstudie wurden nacheinander ausgeführt. Dabei wurde zunächst ein Ausgangsmodell generiert und analysiert. Nach der Beobachtung der neuen Ausbaustufe wurde die Abweichung zur folgenden Ausbaustufe bestimmt und die Generierung und Analyse wiederholt. Dies wurde entsprechend für alle Ausbaustufen durchgeführt. Für die Steuerung wurde eine Managementkomponente verwendet. Für die Analyse wurden die Algorithmen aus Abschnitt 3.5.4 verwendet und Ladiges [Lad18] folgend die Auslastung, der Durchsatz und die Routenflexibilität untersucht.[24] Ladiges gibt in diesem Zusammenhang auch weitere Details der Generierung, die in dieser Arbeit nur verwendet wird.[25]

Machbarkeitsstudie: Nachfolgend wird das Experiment an Beispielen der Ausbaustufen A1 und A2 erläutert. Für **Materialflussmodelle** wird zunächst Ausbaustufe A2 betrachtet, in welcher Werkstücke aus Plastik und Metall durch den Kran vom Magazin zu der Ausgangsrampe transportiert werden.

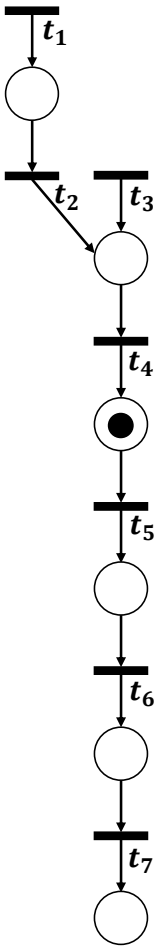
Als für den Materialfluss relevante Ereignistypen besitzt das Magazin die Ereignisse zum Identifizieren der Werkstückausprägung und zum Erkennung der Position am Kran. Weitere Ereignisse sind, den Regeln für Materialflussmodelle folgend, kombinierte Signale des Haltens eines Werkstücks durch den Kran und dessen Position.

Durch die Anwendung des Artefaktverhalten von Materialflussmodellen, die auf der Methode von Ladiges et al. [LFA+15] beruhen, wird mit diesen Ereignistypen das in Abbildung 6.7 gezeigte Materialflussmodell generiert. Die Abbildung zeigt zusätzlich, dem Meta-Modell dieser Arbeit entsprechend, die Kontextinformationen des Modells. Das Modell hat eine Folge von Transitionen, die dem Materialfluss von metallischen Werkstücken entspricht ($t_6 \rightarrow t_7 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5$) und eine Folge für Plastikwerkstücke ($t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5$).

Durch das Beobachten des Zustands über die in der Repräsentationskomponente spezifizierten Aussagen kann die Wissenskomponente auch den aktuellen Zustand durch Marken abbilden. Die in der Abbildung gezeigte Marke wird beispielsweise durch das Feuern der Transition t_3 weitertransportiert, welche abbildet, dass ein Werkstück am Kran hängend an der oberen Position des Magazins angekommen ist. Abweichungen treten dabei z.B. ein, wenn der angegebene Zeitkontext von 125 ms für Plastikwerkstücke signifikant überschritten wird.

[24] Ladiges verwendet in diesem Zusammenhang eigene Algorithmen und Implementierungen zur Analyse, die allerdings in ihrem erzeugten Ergebnis vergleichbar sind.

[25] Ladiges verwendet die selben durch ihn maßgeblich entwickelten Algorithmen zur Generierung, aber eine eigene Implementierungen auf Basis von MatLab



Transition	Signal	Typ	Ort	Zeit
t_1	S.MagazinCapacitive	Sensor-Werkstück-Identifizierung	Magazin	Metall 241 ms
t_2	S.WorkpieceReady	Sensor-Werkstück-Erkennung	Magazin	Metall 2500 ms
t_3	S.WorkpieceReady	Sensor-Werkstück-Erkennung	Magazin	Plastik 2479 ms
t_4	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 120 ms Plastik 119 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnMagazin	Sensor-YPosition	Magazin	
t_5	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 124 ms Plastik 125 ms
	S.Up	Sensor-YPosition	Crane	
	S.CraneOnMagazin	Sensor-YPosition	Magazin	
t_6	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 301 ms Plastik 300 ms
	S.Up	Sensor-YPosition	Crane	
	S. CraneOnConveyor	Sensor-YPosition	Conveyor	
t_7	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 121 ms Plastik 120 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnConveyor	Sensor-YPosition	Conveyor	

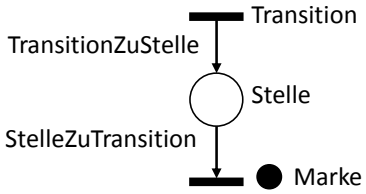
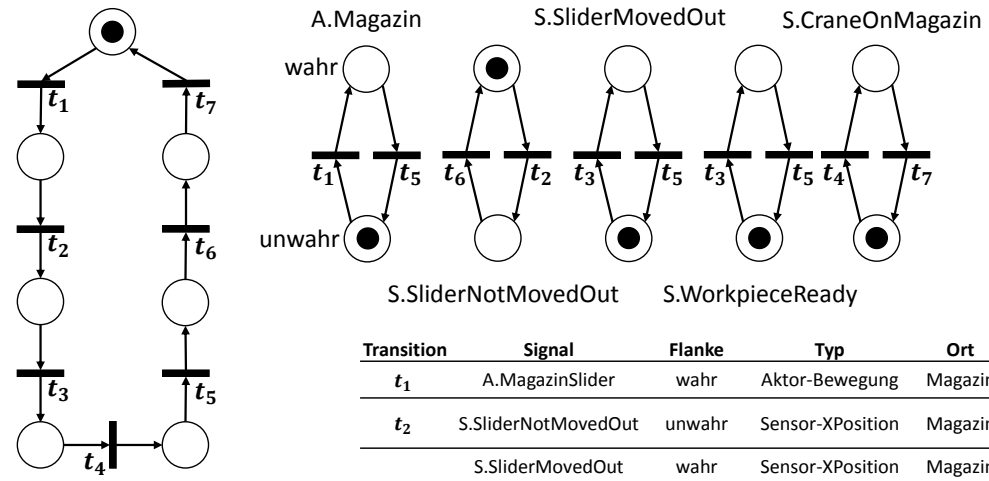
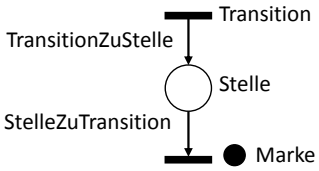


Abbildung 6.7: Materialflussmodell für die Ausbaustufe A2 (ohne Zeitkontext von Stellen)

Maschinenzustandsmodelle werden über eine Wissenskomponente nach der Generierung von Ladiges et al. [LHFL15] für Ereignisse des Magazins erzeugt. Abbildung 6.8 zeigt auf der linken Seite den maschinenbezogenen Teil des generierten Maschinenzustandsmodelle. Das Magazin weist ein zyklisches Verhalten auf indem immer wiederkehrend ein Werkstück an die *WorkpieceReady*-Position transferiert wird. Abbildung 6.8 zeigt zusätzlich auf der rechten Seite die Kontextinformationen des Modells. Das Modell hat dabei auch kombinierte Ereignisse an den Transitionen t_3 und t_5. Dabei haben entgegen der Materialflussmodelle kombinierte Signale an dieser Stelle keine besondere semantische Bedeutung. Sie werden vielmehr kombiniert, da sie im Verhalten des betrachteten Magazins gleichzeitig auftreten.

In Maschinenzustandsmodellen werden beide Flanken eines Signals betrachtet, weshalb dies in der Tabelle vermerkt ist. An dieser Stelle grenzt der signalbasierte Teil des Maschinenzustandsmodells das Verhalten weiter ein. Dieser ist aufgrund des dadurch besseren Verständnisses, entgegen des eigentlichen Maschinenzustandsmodells, in der Abbildung getrennt vom maschinenbasierten Teil notiert. Somit sind die entsprechend mit Namen annotierten Transitionen im eigentlichen Maschinenzustandsmodell identisch. Die Marken beider Teile geben den momentanen Zustand entsprechend der Tabelle 6.7 an.





Transition	Signal	Flanke	Typ	Ort	Zeit
t_1	A.MagazinSlider	wahr	Aktor-Bewegung	Magazin	120 ms
t_2	S.SliderNotMovedOut	unwahr	Sensor-XPosition	Magazin	659 ms
t_3	S.SliderMovedOut	wahr	Sensor-XPosition	Magazin	2221 ms
	S.WorkpieceReady	wahr	Sensor-Werkstück-Erkennung	Magazin	
t_4	S.CraneOnMagazin	wahr	Sensor-XPosition	Magazin	1020 ms
t_5	A.MagazinSlider	unwahr	Aktor-Bewegung	Magazin	138 ms
	S.SliderMovedOut	unwahr	Sensor-XPosition	Magazin	
	S.WorkpieceReady	unwahr	Sensor-Werkstück-Erkennung	Magazin	
t_6	S.SliderNotMovedOut	wahr	Sensor-XPosition	Magazin	2231 ms
t_7	S.CraneOnMagazin	wahr	Sensor-XPosition	Magazin	1890 ms

Abbildung 6.8: Maschinenzustandsmodell für das Magazin in Grundstufe A1

Diskussion der Ergebnisse

Tabelle 6.14 zeigt einen Überblick über die im Experiment generierten Modelle. Zusammenfassend nimmt die Komplexität und damit das Verhalten der PPU-Komponente stetig zu, da es sich in der Mehrzahl der Änderungen um Erweiterungen handelt. Die komplexeste Komponente in seinem Verhalten ist der Kran. Dieser hat schon in Grundstufe A1 34 Transitionen und Stellen.

Exemplarisch zeigt die Tabelle zusätzlich einige nichtfunktionalen Eigenschaften, die aus den CPPS-Modellen berechnet wurden. Die aufgeführte Auslastung wird für jede Komponente aus den Maschinenzustandsmodellen berechnet. Sie ist definiert als das Verhältnis der Zeit, die eine Komponente operativ tätig ist, gegenüber der Zeit, die die Produktion der Werkstücke andauert [LFL16]. Produktionsrate und Routenflexibilität wurden in Abschnitt 3.5.4 bereits definiert.

Aus den koevolvierten Modellen können Rückschlüsse auf die Evolution des beobachteten CPPSs gezogen werden. Für die PPU-Komponente ist beispielsweise exemplarisch festzustellen, dass die ursprüngliche Optimierung von Ausbaustufe A3 zu A4 zwar die Auslastung des Krans erhöht hat (69% $\rightarrow$ 87%), allerdings die Produktionsrate der beiden Werkstücktypen dadurch nicht verbessert worden ist (10s $\rightarrow$ 12s und 15s $\rightarrow$ 22s). Dies resultiert daraus, dass der Kran für die Drehungen zwischen dem Stempel und Magazin deutlich mehr Zeit braucht als die Prägung der Werkstücke durch den Stempel. Entsprechend ist diese Änderung nicht empfehlenswert, da nur die Varianz der Produktionsdauer steigt.

Modell	**Element**	**A1**	**A2**	**A3**	**A4**	**A5**
$MSPN_{Magazin}$	Transition	11	11	13	19	28
$MSPN_{Magazin}$	Stelle	16	16	19	21	27
$MSPN_{Kran}$	Transition	34	34	30	37	45
$MSPN_{Kran}$	Stelle	41	41	40	43	47
$MSPN_{Stempel}$	Transition	-	-	13	16	18
$MSPN_{Stempel}$	Stelle	-	-	28	29	29
$MSPN_{Laufband}$	Transition	-	-	-	-	15
$MSPN_{Laufband}$	Stelle	-	-	-	-	27
MFPN	Transition	5	7	12	12	21
MFPN	Stelle	5	6	11	11	19
$Auslastung_{Magazin}$		0,49	0,49	0,73	0,80	0,92
$Auslastung_{Kran}$		1	1	0,69	0,87	0,85
$Auslastung_{Stempel}$		-	-	0,1	0,37	0,40
$Auslastung_{Laufband}$		-	-	-	-	0,38
$Produktionsrate_{Plastik}$		8,6	8,6	6	5	3,8
$Produktionsrate_{Metall}$		-	8,6	4	2,7	2,1
Routenflexibilität		0	0,5	0,5	0,5	0,83

Tabelle 6.14: Kennzahlen der Modelle für die CPPS-Fallstudie

Folgende allgemeine Schlussfolgerungen ergeben sich durch das Experiment:

- Eine Koevolution von Modellen kann durch die entworfene Methodik erreicht werden. Die zielorientierte Steuerung des Prozesses von Beobachtung, Generierung und Analyse der wissenstragenden Komponente konnte dabei einer halbautomatisierten Koevolution gerecht werden.
- Der halbautomatisierte Ansatz erfordert regelmäßig die Entscheidung des Operators bezüglich der Abweichungen. Im Experiment hat sich gezeigt, dass auch ungewollte Änderungen auftreten, die zumeist auf Fehler in der Laboranlage der Fallstudie zurückzuführen waren. Es zeigt sich dabei, dass eine starke Verbindung zwischen der Fehlererkennung und der Evolutionserkennung vorliegt.
- Für die Fallstudie lag die durchschnittliche Steigerungsrate für Modellelemente über alle betrachteten Modelle und damit die Komplexitätserhöhung des Verhaltens bei 23 %. Das Experiment bleibt allerdings auf eine für die Evolutionsforschung geeignete, aber für Industriesysteme nicht repräsentative Komplexität beschränkt.
- Das Experiment zeigt, dass die Modellebene verbesserte Möglichkeiten zur Bewertung bietet. Beispielweise kann die Optimierung des Kranverhaltens als ungeeignet identifiziert werden.

Einbindung allgemeiner Modellartefakte von CBCPSe

Nachfolgend wird die Koevolution bezüglich der Einbindung von CBCPS-Modellen untersucht.

Experimentaufbau: In diesem Experiment werden die Nachrichtensequenzen genutzt, um Methodenverhaltensmodelle mit ihrer Verbindung zu Repositorymodellen zu koevolvieren. Dafür wird der Experimentaufbau für Dienstaufrufe verwendet. Wissenskomponenten mit Methodenverhaltensmodellen werden dynamisch für jede in den Nachrichtensequenzen auftauchende Methode durch die Managementkomponente erzeugt. Zusätzlich existiert eine Wissenskomponente für das Repositorymodell.

Für die Auswertung der Modelle wird die *Palladio Component Model Workbench*[26] in Version 4.1 verwendet. Um die Simulation der Performanz für die generierten Modelle zu evaluieren, wurde zusätzlich ein *PCM-Assembly-Modell* manuell modelliert, das angibt wie viele Instanzen der Komponenten und Dienste vorhandene sind. Weiterhin wird auch ein manuelles *PCM-Allocation-Modell* verwendet, das beschreibt auf welchen Ressourcen die Komponenten und Dienste laufen. Da ein festes Zeitverhalten der Methoden vorliegt, ist dieses im Rahmen des Experiments nicht weiter relevant. Als letztes manuelles Modell liegt ein *PCM-Usage-Modell* vor, das die Aufrufe an die Hardware und somit den eigentlichen Verkaufsprozess simuliert. Hierfür wurden drei Modelle mit einen durchschnittlichen Verkaufsprozess von 45, 30 und 15 Sekunden genutzt.[27] Die Verkaufsprozesse enthalten dabei durchschnittlich 10 Produkte[28] und mit einer 30-prozentigen Wahrscheinlichkeit wird eine Produktwiedereingabe verwendet.

Machbarkeitsstudie: Exemplarisch wird hier die Generierung für das Repositorymodell und die Methode *KassenApplikation.barcodescanned* gezeigt. Aus der beobachteten Nachrichtensequenz (siehe Tabelle 6.10) und ihren Kontexten (siehe Tabelle 3.2) der vorangegangenen Experimente kann mit Verfahrensvorschrift 1 das in Abbildung 6.9 gezeigte **Repositorymodell** generiert werden.

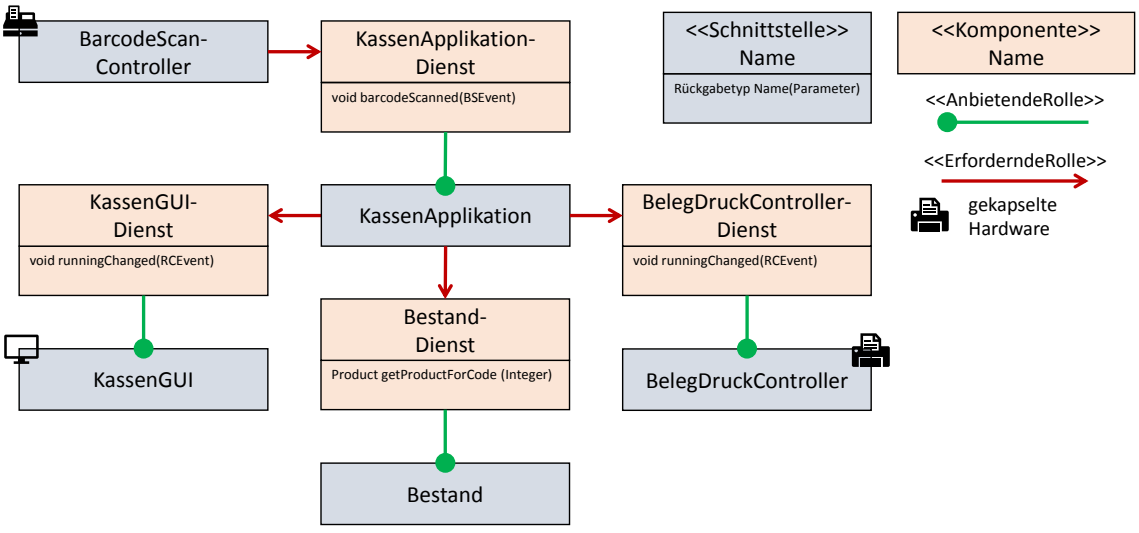


Abbildung 6.9: Repositorymodell von allen durch die Methode *KassenApplikation.barcodescanned* genutzten Relationen

[26] https://sdqweb.ipd.kit.edu/wiki/Palladio_Component_Model
[27] Normalverteilung mit den genannten Erwartungswerten und $\sigma = 1$
[28] Normalverteilung $\mathcal{N}(10,\ 1)$

Abbildung 6.9 folgt dem Meta-Modell von Repositorymodellen und beinhaltet die aufrufenden Komponenten *BarcodeScanController*, *KassenApplikation*, *KassenGUIController*, *Bestand* und *BelegDruckController*. Es werden weiterhin fünf Methoden, die Teil der beobachteten Nachrichtensequenz sind, identifiziert. Es zeigt sich, dass es dabei notwendig ist, auch die Komponenten bzw. deren Methoden im Ortskontext abzubilden, da zwei identische Methoden für die Komponenten *KassenGUIController* und *BelegDruckController* existieren. Von den Methoden hat nur die Anfrage an den *BestandsDienst* einen echten Rückgabewert. Das Modell ist mit nur dieser einen Nachrichtensequenz nicht vollständig, sondern wird, ausgelöst durch Abweichungen bei weiteren auftretenden Nachrichtensequenzen, ständig erweitert.

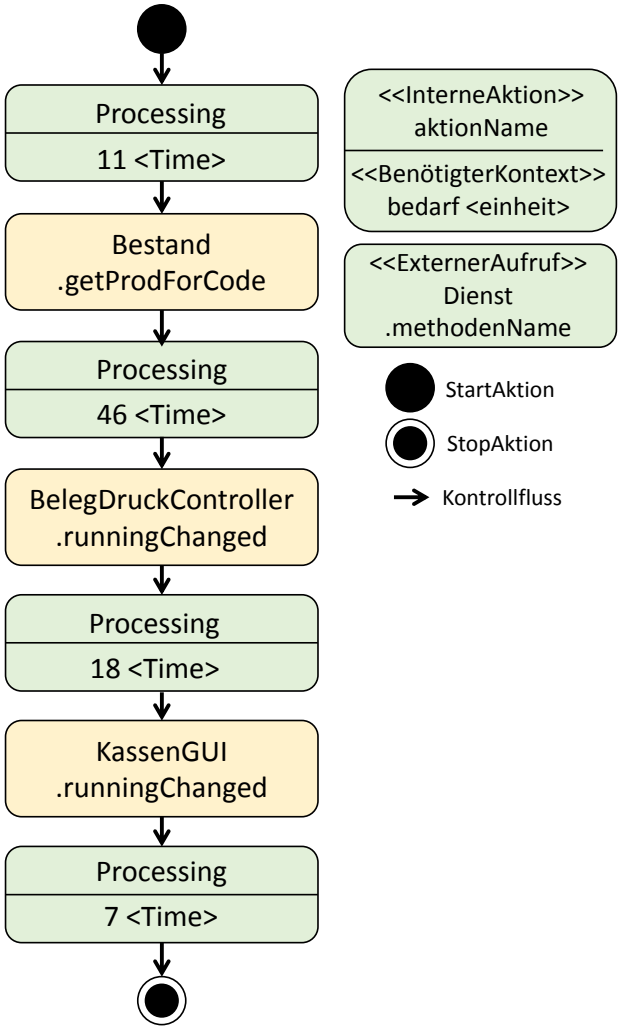


Abbildung 6.10: Methodenverhaltensmodell der Methode *KassenApplikation.barcodescanned*

Abbildung 6.10 zeigt das nach Verfahrensvorschrift 2 genierte **Methodenverhaltensmodell** für die aufgerufene *KA.barcodescanned*-Methode. Wie aus der Nachrichtensequenz (siehe Tabelle 6.10) ersichtlich wird, gibt es als Methoden die Abfrage des Produktinformationen und die Benachrichtigung der Komponenten *BelegDruckerController* und *KassenGUIController*. In der Abbildung sind dem Meta-Modell entsprechend die *InterneAktion*-Elemente enthalten, wobei μ als Zeitbedarf angegeben wird. Alle Dienstaufrufe sind durch ihr *ExterneAktion*-Element repräsentiert.

Diskussion der Ergebnisse

Durch das Experiment konnten ein vollständiges Repositorymodell und in der Grundstufe 14 Methodenverhaltensmodelle extrahiert werden. Beide Modellarten sind miteinander verknüpft und beschreiben die Fallstudie aus Sicht eines Komponentenentwicklers.

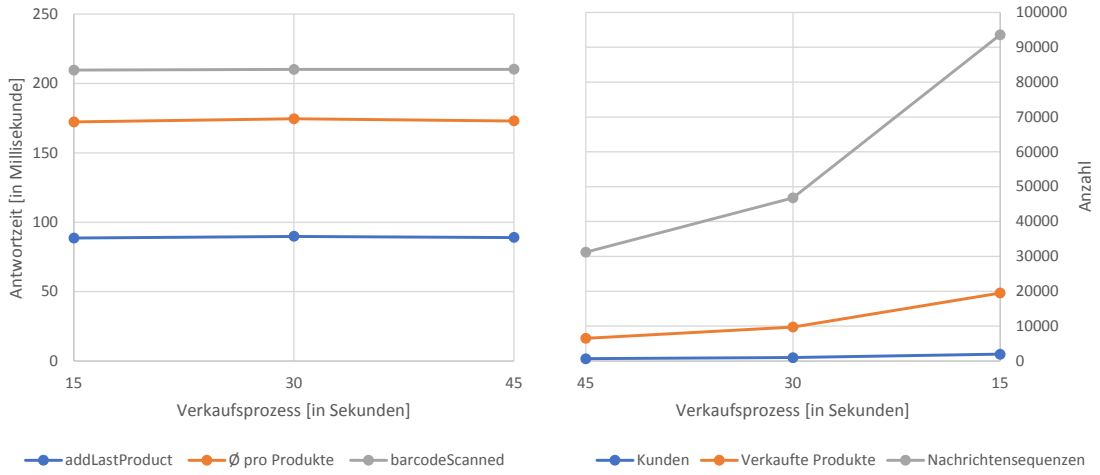


Abbildung 6.11: Antwortzeiten ausgewählter Methoden bei unterschiedlicher Last (links); Anzahl der Kunden, verkaufter Produkte und Methodenaufrufen bei unterschiedlicher Last (rechts)

Mit den generierten Modellen und den im Experimentaufbau genannten zusätzlichen manuellen PCM-Modellen wurde eine Performanzanalyse mit dem PCM-Framework durchgeführt. Die Ergebnisse zeigt Abbildung 6.11. Wie im rechten Diagramm zu sehen ist, steigt die Kundenanzahl und die Anzahl der verkauften Produkte proportional zueinander an. Durch die dadurch ausgelösten Scann-Vorgänge und weiteren Aktivitäten des Systems steigen die simulierten Methodenaufrufe stärker, allerdings weiterhin linear zur Kundenanzahl an.

Wie auf dem linken Diagramm zu sehen, wird dieser Anstieg jedoch nicht in den Antwortzeiten der Methoden widergespiegelt. Diese müssten im Verhältnis zu den Methodenaufrufen und damit den Nachrichtensequenzen ansteigen. Dies hat den Grund, dass die Generierungsverfahren in dem gewählten *Black-Box*-Ansatz keine unterschiedlichen Lasten oder Konfigurationen berücksichtigen. Bei der klassischen Verwendung von PCM wird durch das abgebildete Codeverhalten die Last auf das System über das *PCM-Usage-Modell* berücksichtigt. Entsprechend kann an dieser Stelle in dieser Arbeit nicht der vollständige Funktionsumfang von PCM genutzt werden.

Die folgenden allgemeine Schlussfolgerungen wurden aus diesem Experiment gezogen:

- Eine Generierung von Repositorymodell und Methodenverhaltensmodell ist auch für die softwareintensiveren CBCPSe möglich. Eine Koevolution wird dabei für noch nicht beobachtete Methodenaufrufen, wie in der Ausbaustufe, vorgenommen.
- Die generierten Modelle sind für eine Analyse in PCM einsetzbar, auch wenn durch das gewählte Black-Box-Verfahren keine Berücksichtigung von Last erfolgt. Hier wären alternative Verfahren einzubinden. Diese berücksichtigen häufig Softwarecode nicht nur für eine Kontextanalyse, sondern auch zur Verhaltensextraktion im stetigen Evolutionsprozess (siehe beispielsweise Walter et al. [Wal18, WSKK17]).

- Die Integration von vorhandenem Modelltypen in die wissenstragende Komponente erfordert folgenden Aufwand: Es muss ein Meta-Modell des Modelltyps vorliegen. Für dieses ist in einer Wissenskomponente ein Generierungsverfahren und ein Beobachtungsverfahren zu implementieren. Dazu bedarf es einer entsprechende technische Informationsgrundlage über Adapter- und Repräsentationskomponenten. Dieser Aufwand ist jedoch nicht für jedes CPSs neu zu leisten. In diesem Fall sind Adapterkomponenten ggf. nur neu zu konfigurieren.
- Die in der Fallstudie betrachteten Methoden weisen keine komplexe Geschäftslogik auf, weshalb die Fallstudie für komplexere CBCPS nicht allgemeingültig ist.

Zusammenfassende Bewertung des dritten Evaluationsziels

Die beiden Experimente haben gezeigt, dass eine Koevolution durch die in dieser Arbeit vorgeschlagene Evolutionsunterstützung und wissenstragende Komponente erreicht wird. Dabei konnten bei den gewählten, allerdings beschränkt komplexen Fallstudien gute Evaluationsergebnisse erzielt werden. Es wurde eine Generierung, Analyse und Erkennung von Abweichungen erfolgreich durchgeführt.

Sowohl für hardwarenahe CPPSe als auch arbeitsintensivere CBCPSe konnten Modelltypen, die für CPPSe explizit für die Methodik entwickelt wurden und für CBCPSe etablierten Modelltypen sind, mit vertretbarem Aufwand methodisch und operativ integriert werden.

6.3.4 Evolutionsschritte von Versionen

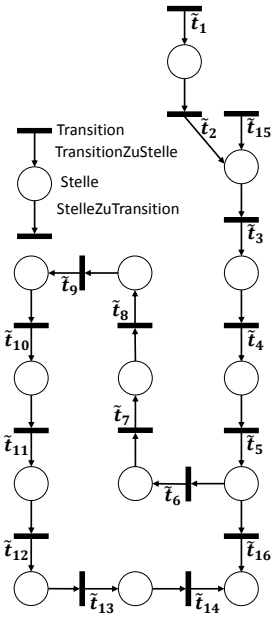
Um eine Unterstützung bei der sequenziellen Evolution durchzuführen, werden nachfolgend Evolutionsschritte von Materialflussmodellen gebildet und zwischen verschiedenen wissenstragenden Komponenten ausgetauscht. Dadurch soll die Unterstützung der Methodik für Versionen evaluiert werden.

Bildung von Evolutionsschritten

Zunächst wird die Bildung von Evolutionsschritten evaluiert.

Experimentaufbau: Die Versionen der Modelle werden entsprechend des Experimentaufbaus der Koevolution für die Grund- und Ausbaustufen der PPU-Komponente erzeugt. Diese Versionen werden softwaretechnisch als dynamische EMF-Modelle in der Evolutionsschrittkomponente verwendet. Eigenschaften mit ihren Differenzen und Abweichungen mit ihren Ereignissen und Kategorien werden als *JavaBeans* verwendet.

Machbarkeitsstudie: Zur Erläuterung des Vorgehens wird zunächst die Bildung des Evolutionsschrittes $\Delta(m_{A2}, \tilde{m}_{A3})$ betrachtet. Der Unterschied der zwei Ausbaustufen liegt hierbei darin, dass ein zusätzlicher Stempel hinzugefügt wird. Abbildung 3.14 zeigte bereits das Materialflussmodell der Ausbaustufe A2. In Abbildung 6.12 wird das Materialflussmodell für Ausbaustufe A3 gezeigt. Das Verhalten des Stempels startet mit $\tilde{t}_6$ und endet mit $\tilde{t}_{12}$.



Transition	Signal	Typ	Ort	Zeit
$\tilde{t}_1$	S.MagazinCapacitive	Sensor-Werkstück-Identifizierung	Magazin	Metall 241 ms
$\tilde{t}_2$	S.WorkpieceReady	Sensor-Werkstück-Erkennung	Magazin	Metall 2500 ms
$\tilde{t}_3$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 121 ms Plastik 119 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnMagazin	Sensor-XPosition	Magazin	
$\tilde{t}_4$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 123 ms Plastik 125 ms
	S.Up	Sensor-YPosition	Crane	
	S.CraneOnMagazin	Sensor-XPosition	Magazin	
$\tilde{t}_5$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 120 ms Plastik 300 ms
	S.Up	Sensor-YPosition	Crane	
	S. CraneOnConveyor	Sensor-XPosition	Conveyor	
$\tilde{t}_6$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 301 ms
	S.Up	Sensor-YPosition	Crane	
	S.CraneOnStamp	Sensor-XPosition	Stamp	
$\tilde{t}_7$	S.SliderFilled	Sensor-Werkstück-Erkennung	Stamp	Metall 602 ms
	S.MovedOut	Aktor-Move	Stamp	
$\tilde{t}_8$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 60 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnStamp	Sensor-XPosition	Stamp	

Transition	Signal	Typ	Ort	Zeit
$\tilde{t}_9$	S.SliderFilled	Sensor-Werkstück-Erkennung	Stamp	Metall 929 ms
	A.MovedIn	Aktor-Move	Stamp	
$\tilde{t}_{10}$	S.SliderFilled	Sensor-Werkstück-Erkennung	Stamp	Metall 2368 ms
	A.MovedOut	Aktor-Move	Stamp	
$\tilde{t}_{11}$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 119 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnStamp	Sensor-XPosition	Stamp	
$\tilde{t}_{12}$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 181 ms
	S.Up	Sensor-YPosition	Crane	
	S.CraneOnStamp	Sensor-XPosition	Stamp	
$\tilde{t}_{13}$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 302 ms
	S.Up	Sensor-YPosition	Crane	
	S. CraneOnConveyor	Sensor-XPosition	Conveyor	
$\tilde{t}_{14}$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 120 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnConveyor	Sensor-XPosition	Conveyor	
$\tilde{t}_{15}$	S.WorkpieceReady	Sensor-Werkstück-Erkennung	Magazin	Plastik 2220 ms
$\tilde{t}_{16}$	S.Sucked	Aktor-Werkstück-Halten	Crane	Plastik 121 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnConveyor	Sensor-XPosition	Conveyor	

Abbildung 6.12: Materialflussmodell der Ausbaustufe A3

Zuordnung: Für die Ableitung eines Evolutionsschrittes ist eine Zuordnung von Modellen nach Abschnitt 3.5.2 notwendig. Für Ausbaustufe A2 und A3 wird diese nun anhand der Kontexte der Zuordnung $t_2 \rightarrow \tilde{t}_5$ beispielhaft für das Experiment gezeigt:

- **Zeitkontext:** Unter den Mittelwerte $\mu = 120$ und $\tilde{\mu} = 122$ ergibt sich für den Zweistichproben-t-Test der Standardfehler 2,41 für metallische Werkstücke.[29] Somit lässt sich die Ähnlichkeit unter $\Theta_{minZeit} = 100$ für den metallische Zeitkontext berechnen:

$$ähnlichkeit_{\tilde{z}_{k_1}} = 1 - \frac{2,41}{100} = 0,9759 \tag{6.1}$$

  Da für Plastikwerkstücke analog $\tilde{z}_{k_2} = 218$ berechnet wird, ergibt sich die Ähnlichkeit des Zeitkontextes:

$$ähnlichkeit^{Zeitkontext}_{t_2 \leftrightarrow \tilde{t}_5} = \frac{0,9759 + 0}{2} = 0,488 \tag{6.2}$$

- **Ortkontext:** Für den Ortskontext sind die Vektoren $\{x_{magazin}, x_{crane}, x_{stamp}, x_{conveyor}\} = \{1, 2, 0, 0\}$ von t_2, sowie $\{0, 2, 1, 0\}$ von $\tilde{t}_5$ zu vergleichen. Damit kann eine Ähnlichkeit des Ortskontextes berechnet werden:

$$ähnlichkeit^{Ortskontext}_{t_2 \leftrightarrow \tilde{t}_5} = \frac{0 + 2 + 0 + 0}{0 + 2 + 1 + 1} = 0,5 \tag{6.3}$$

- **Typkontext:** Neben den identischen Relationen besteht beim Ortskontext eine Relation zwischen $S.XPosition \leftrightarrow S.YPosition = 0,75$. Darunter ergibt sich unter Einhaltung der Nebenbedingungen die Lösung des Zuordnungsproblems vom Ortskontext:

$$\begin{array}{l} \\ \textbf{A.Hold} \\ \textbf{S.XPosition} \\ \textbf{S.YPosition} \end{array} \begin{array}{c} \begin{array}{ccc} \textbf{A.Hold} & \textbf{S.XPosition} & \textbf{S.YPosition} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0,75 \\ 0 & 0,75 & 1 \end{pmatrix} \end{array} \tag{6.4}$$

  Entsprechend ist die Ähnlichkeit des Typkontextes maximal:

$$ähnlichkeit^{Typkontext}_{t_2 \leftrightarrow \tilde{t}_5} = \frac{2 \cdot (1 \cdot 1 + 1 \cdot 1 + 1) \cdot 1}{3} = 1 \tag{6.5}$$

- Unter einem gleichgewichteten Mittel wird dann die Gesamtähnlichkeiten berechnet. Diese zeigt für alle Paare von Transitionen Tabelle 6.15. Die betrachtete Zuordnung $t_2 \rightarrow \tilde{t}_5$ ist mit $0,67$ rot markiert.

Nachfolgend wird mit den Ähnlichkeiten der einzelnen Modellelemente eine Anordnung für die zwei metallischen Folgen $t_6, t_7, t_1, t_2, t_3, t_4, t_5$, sowie $\tilde{t}_1, \tilde{t}_2, \cdots, \tilde{t}_{14}$ bestimmt.

[29] Weitere Werte: $\sigma^2 = 1,71$, $\sigma^2 = 2,40$, $n = 6$, $\tilde{n} = 6$

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
$\tilde{t}_1$	0,58	0,15	0,15	0,09	0,04	1	0,58
$\tilde{t}_2$	0,67	0,15	0,15	0,04	0,04	0,58	1
$\tilde{t}_3$	0,15	1	0,99	0,5	0,83	0,15	0,15
$\tilde{t}_4$	0,15	0,99	0,99	0,5	0,83	0,15	0,15
$\tilde{t}_5$	0,04	0,67	0,67	0,83	0,83	0,04	0,04
$\tilde{t}_6$	0,04	0,83	0,83	0,5	0,83	0,04	0,04
$\tilde{t}_7$	0,22	0,03	0,03	0,03	0,03	0,17	0,22
$\tilde{t}_8$	0,04	0,55	0,55	0,5	0,54	0,04	0
$\tilde{t}_9$	0,22	0,03	0,03	0,03	0,03	0,17	0,22
$\tilde{t}_{10}$	0,22	0,03	0,03	0,03	0,03	0,17	0,22
$\tilde{t}_{11}$	0,04	0,83	0,83	0,33	0,66	0,04	0,04
$\tilde{t}_{12}$	0,04	0,54	0,54	0,33	0,38	0,09	0,04
$\tilde{t}_{13}$	0,04	0,5	0,5	1	0,67	0,09	0,04
$\tilde{t}_{14}$	0,04	0,83	0,83	0,67	1	0,04	0,04
$\tilde{t}_{15}$	0,67	0,11	0,11	0	0	0,58	0,67
$\tilde{t}_{16}$	0,04	0,82	0,83	0,67	1	0,04	0,04

Tabelle 6.15: Ähnlichkeiten der Transition von Ausbaustufe A2 und A3

Dazu wird die Substitutionsmatrix gebildet[30] und die Verfahrensvorschrift 3 durchgeführt[31]. Diese ergibt die in der Formel 6.6 gegebene Wertungsmatrix der zwei betrachteten Folgen von m und $\tilde{m}$. Durch die Rückverfolgung ergibt sich die rot markierte optimale Lösung. Alle in dieser Lösung diagonal zuordneten Transitionen sind als Zuordnung Teil der optimalen Anordnung der zwei Folgen.

$$
\begin{array}{c}
 \\
Init \\
\tilde{t}_1 \\
\tilde{t}_2 \\
\tilde{t}_3 \\
\tilde{t}_4 \\
\tilde{t}_5 \\
\tilde{t}_6 \\
\tilde{t}_7 \\
\tilde{t}_8 \\
\tilde{t}_9 \\
\tilde{t}_{10} \\
\tilde{t}_{11} \\
\tilde{t}_{12} \\
\tilde{t}_{13} \\
\tilde{t}_{14}
\end{array}
\begin{array}{c}
\begin{array}{ccccccc}
Init & t_6 & t_7 & t_2 & t_3 & t_4 & t_5
\end{array} \\
\left(\begin{array}{ccccccc}
0 & -8 & -10 & -12 & -14 & -16 & -18 \\
-8 & 10 & 15,8 & 5,8 & -4,2 & -14,2 & -24,2 \\
-10 & 0 & 20 & 10 & 0 & -10 & -20 \\
-12 & -2 & 10 & 30 & 20 & 10 & 0 \\
-14 & -4 & 8 & 20 & 39,9 & 29,9 & 19,9 \\
-16 & -6 & 6 & 18 & 29,9 & 48,2 & 38,2 \\
-18 & -8 & 4 & 16 & 27,9 & 38,2 & 56,5 \\
-20 & -10 & 2 & 14 & 25,9 & 36,2 & 46,5 \\
-22 & -12 & 0 & 12 & 23,9 & 34,2 & 44,5 \\
-24 & -14 & -2 & 10 & 21,9 & 32,2 & 42,5 \\
-26 & -16 & -4 & 8 & 19,9 & 30,2 & 40,5 \\
-28 & -18 & -6 & 6 & 17,9 & 28,2 & 38,5 \\
-30 & -20 & -8 & 4 & 15,9 & 26,2 & 36,5 \\
-32 & -22 & -10 & 2 & 13,9 & 25,9 & 34,5 \\
-34 & -24 & -12 & 0 & 11,9 & 20,6 & 35,9
\end{array}\right)
\end{array}
\tag{6.6}
$$

[30] Es wurde $\Theta_{lokal} = 0,5, k_{sw} = 10$ verwendet.

[31] Es wurde der Strafwert für Lücken $l = 10$, was einem vollständig ähnlicher Transition entspricht, und der Strafwert für Lückenverlängerungen $lv = 2$ verwendet.

Beide Materialflussmodelle besitzen neben einer metallischen Folge auch genau eine Folge für Plastikwerkstücke. Für diese ergibt sich nach dem Zuordnungsverfahren folgende Anordnung:

t_1	t_2	t_3	t_4	t_5
$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$
$\tilde{t}_{15}$	$\tilde{t}_3$	$\tilde{t}_4$	$\tilde{t}_5$	$\tilde{t}_{16}$

Tabelle 6.16: Anordnung der Folgen für Plastikwerkstücke von Ausbaustufe A2 und A3

Unter der Zuordnungsmatrix, welche die Anzahl der Zuordnungen der indizierten Modellelemente $z(t_i, \tilde{t}_{\tilde{i}})$ in allen Anordnungen als Einträge besitzt, hat das endgültige Zuordnungsproblem (siehe Formel 3.23) zwei gleichwertige Lösungen. Denn für die Transitionen t_4 und t_5 können nach den metallischen Werkstücken die Transitionen $\tilde{t}_{13}$ und $\tilde{t}_{14}$ zugewiesen (siehe Matrix 6.6) werden als auch gleichwertig nach den Plastikwerkstücken die Transitionen $\tilde{t}_5$ und $\tilde{t}_{16}$ (siehe Tabelle 6.16). An dieser Stelle werden die Konfliktauflösungsregeln verwendet. Im betrachteten Fall ist dabei die längere zusammenhängende Folge von Zuordnungen bei Plastikwerkstücken ausschlaggebend (Regel 1.). Deshalb ergibt sich nach der Konfliktlösung folgende endgültige Anordnung:

Zuordnungen							**Ohne Zuordnung**								
t_1	t_2	t_3	t_4	t_5	t_6	t_7	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$	$\updownarrow$									
$\tilde{t}_{15}$	$\tilde{t}_3$	$\tilde{t}_4$	$\tilde{t}_5$	$\tilde{t}_{16}$	$\tilde{t}_1$	$\tilde{t}_2$	$\tilde{t}_6$	$\tilde{t}_7$	$\tilde{t}_8$	$\tilde{t}_9$	$\tilde{t}_{10}$	$\tilde{t}_{11}$	$\tilde{t}_{12}$	$\tilde{t}_{13}$	$\tilde{t}_{14}$

Tabelle 6.17: Zuordnung der Transitionen der zwei Materialflussmodelle von Ausbaustufe A2 und A3

Dabei bleiben die Transitionen des Stempels ohne Zuordnung, was im Sinne der Reaktion bedeutet, dass diese während der Evolution erzeugt wurden.

Grund: Im Grund des Evolutionsschrittes $\Delta(m_{A2}, \tilde{m}_{A3})$ treten folgende Abweichungen auf (vergleiche Kategorisierung in Abschnitt 3.5.5):

- **Wegfallen:** Es wird im Materialflussmodell nicht mehr beobachtet, dass metallische Werkstücke direkt über Transition t_5 zur Ausgabe transportiert werden.
- **Einfügung:** Es wird nun allerdings beobachtet, dass metallische Werkstücke an dieser Stelle weiter zum Stempel transportiert werden - was im später generierten Modell durch Transition $\tilde{t}_5$ abgebildet wird.
- **Zeitaberration:** Eine zeitliche Differenz tritt beim Warten der Werkstücke auf den Kran bei $\tilde{t}_2$ und $\tilde{t}_{15}$ auf. Zusätzlich am kombinierten Ereignis $(S.Sucked,\ S.Up,\ S.CraneInConveyor)$ von $\tilde{t}_2$ für metallische Werkstücke, da der Kran nicht mehr wie in Ausbaustufe A2 für metallische Werkstücke stoppt und somit der Wert des Ereignisses länger im Zustand gültig ist.
- **Übergreifend:** Die Änderung, dass keine metallischen Werkstücke mehr direkt zur Rampe transportiert werden, wird auch dadurch erkannt, dass die Anzahl der Werkstücke n an Transition t_5 sich nicht mehr verändert.

Bei der in der Fallstudie gewählten Konfiguration tritt die Einfügung der Kranereignisse von $\tilde{t}_6$ als erste Abweichung auf. Diese Abweichung bildet mit der Klasse der Abweichung und dem erwarteten und dem beobachteten Ereignis das TriggerVon-Wertetypobjekt des Grundartefakt.

Reaktion: Nachfolgend wird eine ausschnittsweise Berechnung des automatisierten Teils einer Wirkung der Reaktion gezeigt. Dazu werden die Ausbaustufen A4 und A5 betrachtet.

Abbildung 6.13 zeigt einen Ausschnitt des abstrakten Syntaxgraphen des Ursprungsmodells m von Ausbaustufe A4 und des Ursprungsmodells $\tilde{m}$ der Ausbaustufe A5. Im grau hinterlegten Teil der Abbildung wird zusätzlich der entsprechende Ausschnitt der Modelle gezeigt.

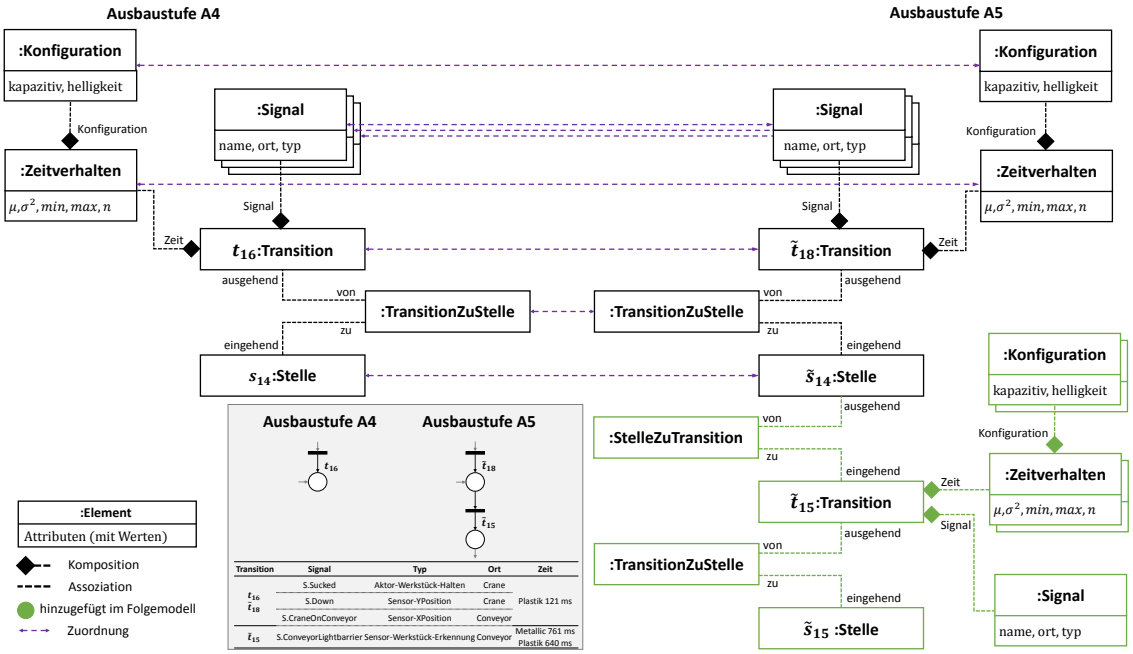


Abbildung 6.13: Ausschnitt des abstrakten Syntaxgraphen von Ausbaustufe A4 und A5 mit Zuordnungen, Markierungen der hinzugefügten Elemente und den Modellinstanzen

Die dort gezeigte Transition t_{16} bildet ein kombiniertes Ereignis für Plastikwerkstücke ab. Es beschreibt das Loslassen des Krans an der Ausgangsrampe bzw. dem Laufband. Im Fall der Rampe ist das Modell danach mit einer Stelle abgeschlossen. In Ausbaustufe A5 sind eine Transition $\tilde{t}_{18}$ und eine Stelle mit dem gleichen kombinierten Ereignis vorhanden. Der Stelle folgt nun allerdings die Transitionsfolge des Transportbandes, die mit einer Transition $\tilde{t}_{15}$ ein Signal der neuen Lichtschranke für beide Ausprägungen von Werkstücken beschreibt.

Der abstrakte Syntaxgraph zeigt, dass das Element der Transition t_{16} mit einem Zeitverhalten, das eine Konfiguration für Plastikwerkstücke besitzt, und den drei Signal-Elementen über die entsprechenden Referenzen (Signal, Zeit und Konfiguration) verbunden ist. Der Kontrollfluss zur Stelle s_{14} ist dem Meta-Modell entsprechend über eine TranstionZuStelle-Verbindung mit zweiseitigen Referenzen abgebildet. Alle diese Modellelementen wurden durch das Zuordnungsverfahren den entsprechenden Modellelementen von Transition $\tilde{t}_{18}$ zugeordnet. Die weiteren Modellelemente von Ausbaustufe A5 haben keine Zuordnung im Ursprungsmodell.[32]

[32]Attribute sowie die Referenzen (Element, Verbindung, Netz) der Verbindungen und Elemente zum übergeordneten Modellelement des Materialfluss wurden in dieser Darstellung aufgrund der Lesbarkeit vernachlässigt, sind allerdings in den Modellen gegeben.

Durch das Differenzverfahren kann nun aus beiden Modellen eine Differenz der konzeptionellen Struktur gebildet werden. Daraus ergibt sich eine Differenz mit einer großen Anzahl an primitiven Änderungen[33]. Ausschnittsweise sind diese für die Transition $\tilde{t}_{15}$ und das Verbindungselement $StelleZuTransition$ nachfolgend angegeben:[34]:

1. $erzeugeElement(\tilde{t}_{15})$
2. $erzeugeReference(Netz, \tilde{t}_{15}, MaterialFluss)$
3. $erzeugeReference(Element, MaterialFluss, \tilde{t}_{15})$
4. $erzeugeElement(StelleZuTransition)$
5. $erzeugeReference(Netz, StelleZuTransition, Materialfluss)$
6. $erzeugeReference(Verbindung, Materialfluss, StelleZuTransition)$
7. $erzeugeReference(ausgehend, \tilde{t}_{15}, StelleZuTransition)$
8. $erzeugeReference(zu, StelleZuTransition, \tilde{t}_{15})$

Die primitiven Änderungen umfassen das Erzeugung der Elemente Transition (1.) und StelleZuTransition (4.), sowie deren Verbindung zum Element des Materialflussmodell (2.+4. und 5.+6.) und die in Abbildung 6.13 zu sehenden Assoziationen.

Auf diesen primitiven Änderungen wird nach den in Abschnitt 3.5.1 definierten Operationen für Materialflussmodelle gesucht. Dabei zeigt Abbildung 6.14 mit Operation A (Folgetransition) eine dieser Operationen. Die stereotypische Annotation *erhalten* kennzeichnet Elemente und Referenzen, welche im vorliegenden Ursprungsmodell, sowie im Folgemodell vorliegen müssen. Mit der stereotypischen Annotation *hinzufügen* sind die primitiven Änderungen angegeben, welche vorliegen müssen, damit eine Anwendung der Operation angenommen werden kann.

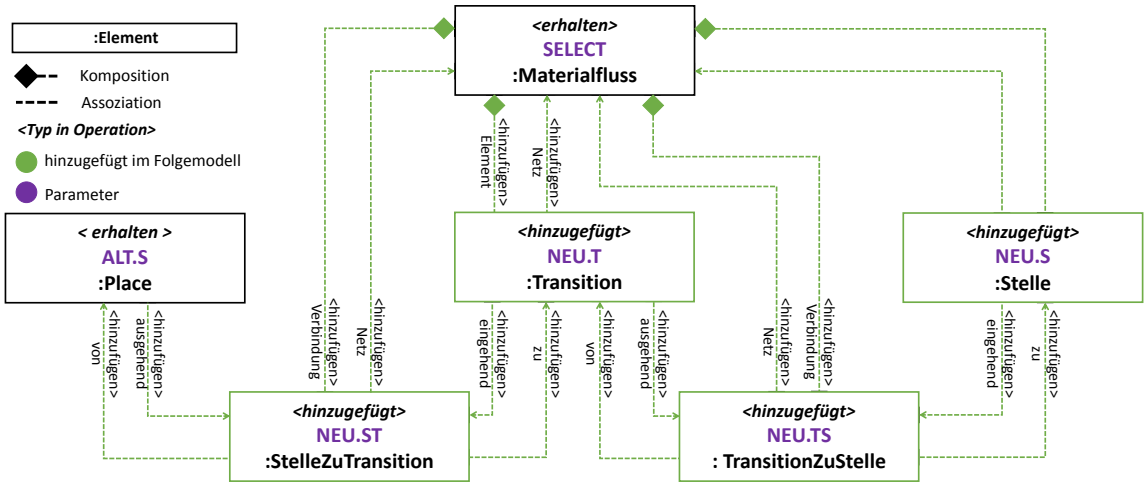


Abbildung 6.14: Die Operation F: Folgetransition

Beim Vergleich der Operation A und dem abstrakten Syntaxgraphen bzw. den primitiven Änderungen zeigt sich, dass es sich um eine Anwendung dieser Operation handelt. Denn sowohl die Voraussetzungen der erhaltenden Elemente als auch die hinzugefügten primitiven Änderungen liegen hier vor. Der abstrakte Syntaxgraph lässt sich somit als Anwendung der Operation A abbilden. Zusätzlich wird zwei Mal die Operation F eines neuen Zeitverhaltens und einmal die Operation E eines neuen Signals anwendet.

[33] Die Arbeit unterschiedet nicht zwischen Änderungen und Pseudoänderungen (*pseudo changes*), also Änderungen die inhaltlich unbedeutend oder redundant sind (vergleiche [Kel10]).
[34] Elemente ohne Namen werden wie Referenzen durch ihren Typ angegeben.

Das daraus resultierende Editierskript $\delta(m, \tilde{m})$ des gezeigten Ausschnitts besteht aus den vier angewendeten Operationen. Für jede Operation ist nachfolgend der Name, sowie die Eingangsparameter (in) und Ausgangsparameter (out) angegeben:

op_1 $Folgetransition($ **in:** $SELECT = Materialfluss,\ ALT.S = s_{14} \rightarrow \tilde{s}_{14}$;
out: $NEU.T = \tilde{t}_{15},\ NEU.P = \tilde{s}_{15},\ NEU.ST = StelleZuTransition,$
$NEU.TS = StelleZuTransition)$

op_2 $Zeitverhalten($ **in:** $SELECT = \tilde{t}_{15}$;
out: $NEU.Z = Zeitverhalten,\ MEAN = 640,\ STD = 28,\ MAX = 661,\ MIN = 600,\ N = 9,\ NEU.K = Konfiguration,\ X = 0,\ Y = 0$

op_3 $Zeitverhalten($ **in:** $SELECT = \tilde{t}_{15}$;
out: $NEU.Z = Zeitverhalten,\ MEAN = 761,\ STD = 28,\ MAX = 781,\ MIN = 719,\ N = 9,\ NEU.K = Konfiguration,\ X = 1,\ Y = 1)$

op_4 $Signal($ **in:** $SELECT = \tilde{t}_{15}$;
out: $NEU.S = Signal,\ name = S.ConveyorLightbarrier,$
$ORT = CONVEYOR,\ TYP = SWPDETECT)$

Zusätzlich dazu werden die Abhängigkeiten der Operationen bestimmt. Diese sind in diesem Fall dadurch gegeben, dass op_1 zuerst ausgeführt werden muss, da die Transition für die späteren Operationen als Eingangsparameter genutzt wird:

$op_1 \prec op_2$
$op_1 \prec op_3$
$op_1 \prec op_4$

Der zur Wirkung gehörende Modellkontext ist der Kontext von s_{14}. Denn diese Stelle wird als alleiniger nicht erzeugter Eingangsparameter genutzt und es wird bestimmt, wo die Operation A angewendet werden kann bzw. sollte. Dabei umfasst der Kontext von Stellen immer die Kontexte seiner jeweils vorherigen und nachfolgenden Transitionen. Dies sind im gezeigten Beispiel $\tilde{t}_{18}$ und $\tilde{t}_{15}$.

Das Editierskript und der Modellkontext stellt das WirkungAn-Wertetypobjekt des Reaktionsartefakts dieses Beispiels dar.

Ergebnis: Um die Ereignisbestimmung zu demonstrieren, werden dieselben Ausbaustufen A4 und A5 verwendet. Abbildung 6.15 zeigt den Ausschnitt des Laufbands für des Materialflussmodell $\tilde{m}$.

Der Ausschnitt zeigt, dass im Laufband, hier für metallisch Werkstücke rot markiert, drei Routen existieren. Diese werden durch die Verfahrensvorschrift 4 mit einer Tiefensuche gefunden. Durch die iterativ ausgeführte Methode BESUCHE($e,\ ausprägung$) ergibt sich durch die sukzessiv gebildete Summe die mittlere Produktionsdauer. Diese entspricht der Summe aller Zeitbedarfe μ und ist für Route 1 entsprechend $120 - 301 + 761 + 1140 + 120 = 1840\ ms$.

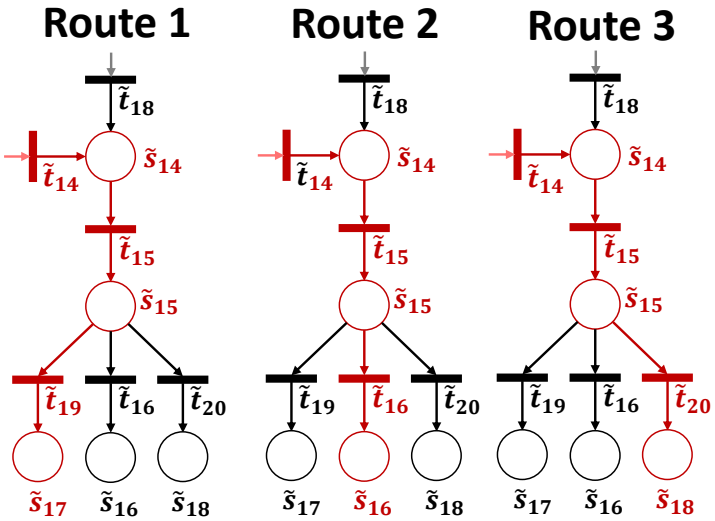
Unter dem gewichteten Mittel der mittleren Produktionsdauer aller Routen ergibt sich die Produktionsrate:[35]

$$f_{analyse}^{Produktionsrate} = \frac{1}{\frac{3261 \cdot 6 + 3260,33 \cdot 6}{12}} = \frac{1}{3260,67} = 5,43\ Werkstücke\ pro\ Min. \tag{6.7}$$

Die Routenflexibilität ist entsprechend:

$$f_{analyse}^{Routenflexibilität} = 1 - \frac{1}{3+3} = 0,83 \tag{6.8}$$

[35] Es wurde ein 1:1-Mix der Werkstoffausprägungen mit gleichmäßiger Sortierung in den Ausgangsrampen angenommen



Transition	Signal	Typ	Ort	Zeit
$\tilde{t}_{14}$	S.Sucked	Aktor-Werkstück-Halten	Crane	Metall 120 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnConveyor	Sensor-XPosition	Conveyor	
$\tilde{t}_{15}$	S.Conveyor-Lightbarrier	Sensor-Werkstück-Erkennung	Conveyor	Metallic 761 ms Plastik 640 ms
$\tilde{t}_{16}$	S.PositionRamp3	Sensor-Werkstück-Erkennung	Conveyor	Metallic 420 ms Plastik 419 ms
$\tilde{t}_{18}$	S.Sucked	Aktor-Werkstück-Halten	Crane	Plastik 121 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnConveyor	Sensor-XPosition	Conveyor	
$\tilde{t}_{19}$	S.SorterPusher1-MovedOut	Sensor-Werkstück-Bewegen	Conveyor	Metallic 120 ms Plastik 121 ms
$\tilde{t}_{20}$	S.SorterPusher2-MovedOut	Sensor-Werkstück-Bewegen	Conveyor	Metallic 120 ms Plastik 120 ms

Stelle	Von/Zu	Zeit
$\tilde{s}_{14}$	$\tilde{t}_{14}$	Metall -301 ms
	$\tilde{t}_{18}$	Plastik -300 ms
$\tilde{s}_{15}$	$\tilde{t}_{16}$	Metallic 4020 ms Plastik 4140 ms
	$\tilde{t}_{19}$	Metallic 1140 ms Plastik 1260 ms
	$\tilde{t}_{20}$	Metallic 2221 ms Plastik 2340 ms
$\tilde{s}_{16}$	-	End
$\tilde{s}_{17}$	-	End
$\tilde{s}_{18}$	-	End

Abbildung 6.15: Routen im Ausschnitt des Laufbands vom Materialflussmodell der Ausbaustufe A5

Durch die Berechnung der Ereignisdifferenzen (siehe Formeln 3.29 und 3.30) ergeben sich die in Tabelle 6.18 angeben Eigenschaftswerte und ihre Ereignisdifferenzen. Es zeigt sich eine Verringerung von $-21,6\%$ in der Produktionsrate und eine erhebliche Verbesserung von 66% in der Routenflexibilität.

		Routenflexiblität	Produktionsrate	Routenflexiblität	Produktionsrate	Routenflexiblität	Produktionsrate
Quelle	**Model**	**Werte**		**AbsDiff**		**RelDiff**	
Berechnung	A4	$0,5$	$3,86$	$0,33$	$-0,86$	$66,66\%$	$-21,6\%$
	A5	$0,83$	$3,03$				

Tabelle 6.18: Eigenschaften des Effekts für Ausbaustufe A4 und A5

Diskussion der Ergebnisse

Nachfolgende werden die gesamten Experimentergebnisse entlang des Grunds, der Reaktion und des Ergebnisses diskutiert.

Über den **Grund** soll erkannt werden, welche Art von Abweichung den Evolutionsschritt ausgelöst hat. Die identifizierten Abweichungen von Materialflussmodellen der Fallstudie zeigt Tabelle 6.19. Diese gibt die Summe der bei einer Beobachtung der Folgeausbaustufe mit dem Modell der vorherigen Ausbaustufe aufgetretenen Abweichungen an.[36] Die rot markierte Zeile entspricht der Klasse der Abweichung, die im Produktionsprozess als erstes auftritt.

Klasse	$\Delta(\emptyset, \tilde{m}_{A1})$	$\Delta(m_{A1}, \tilde{m}_{A2})$	$\Delta(m_{A2}, \tilde{m}_{A3})$	$\Delta(m_{A3}, \tilde{m}_{A4})$	$\Delta(m_{A4}, \tilde{m}_{A5})$
Wegfallen	0	0	1	0	0
Einfügung	**5**	**2**	**9**	0	**4**
Veränderung	0	0	0	0	0
Zeitaberration	0	1	3	**4**	0
Übergreifend	0	0	1	0	0

Tabelle 6.19: Abweichungen der Evolutionsschritte der Fallstudie für CPPSe

Die Kategorisierung zeigt, dass in den Ausbaustufen insbesondere Ereignisse eingefügt werden. Dies resultiert aus der sukzessive Komplexitätserhöhung der Fallstudie. Die Klasse *Einfügung* wurde im Experiment sehr zuverlässig erkannt und konnte in der Fallstudie als wichtigster Indikator für Evolution identifiziert werden.

Ein *Wegfall* von Ereignissen ist nur beim direkten Transport von metallischen Werkstücken zur Ausgangsrampe aufgetreten, da dieses Verhalten in den späteren Ausbaustufen nicht mehr stattfindet. An dieser Stelle treten am häufigsten Abweichungen durch ungewollte Fehler der PPU-Komponenten auf. Dennoch wurden fehlende Ereignisse zuverlässig erkannt.

Das Wegfallen von metallischen Werkstücken beim direkten Transport wird auch durch die modellspezifischen Abweichungen anhand der Anzahl von Werkstücken erkannt. Solche Abweichungen der Klasse *Übergreifend* wurden in Materialflussmodellen nur für diesen Fall modelliert und sind nach Erkenntnissen des Experimentes systemspezifisch zu modellieren. Abweichungen der Klasse *Veränderungen* traten in Materialflussmodellen nicht auf, wobei diese vor allen beim Austausch von technischen Elementen, wie Sensoren, auftreten sollten. Durch die vermehrte Kombination von zeitgleichen Signalen sind diese allerdings in Maschinenzustandsmodellen zu beobachten, auch wenn deren Erkennung als schwierig zu bewerten eingestuft wurden.

Zeitaberrationen treten als Grund der Evolution mehrfach auf. Im Fall der Optimierung des Kranverhaltens war dies auch die ausschlaggebende erste Abweichung. Die Abweichungen sind insbesondere an den Stellen signifikant, an denen auf die Produktion anderer Werkstücke gewartet wird. Allerdings traten grade an diesen Stellen auch Zeitaberrationen durch statistische Schwankungen auf. Diese Erkennung ist deshalb als konfigurationsabhängig zu betrachten, wobei das entstehende Spannungsfeld zwischen falsch-positiven und falsch-negativen Abweichungen durch den Nutzer aufzulösen ist.

[36] Bei der Grundstufe wird ein leeres Modell verwendet.

Das **Ergebnis** dient dazu, den Einfluss auf nichtfunktionale Eigenschaften des Evolutionsschrittes abzuschätzen. Tabelle 6.20 zeigt die im Experiment berechneten Eigenschaften.

Eigenschaft	**Typ**	$\Delta(\emptyset, A1)$	$\Delta(m_{A1}, \tilde{m}_{A2})$	$\Delta(m_{A2}, \tilde{m}_{A3})$	$\Delta(m_{A3}, \tilde{m}_{A4})$	$\Delta(m_{A4}, \tilde{m}_{A5})$
Routenflexibilität	AbsDiff	0	$0,5$	0	0	$0,33$
	RelDiff	0	∞	0	0	66%
$\text{Produktionsrate}_{\text{Plastik}}$	AbsDiff	$8,6$	0	$-2,6$	-1	$-1,25$
	RelDiff	∞	0%	-30%	$-16,7\%$	-25%
$\text{Produktionsrate}_{\text{Metall}}$	AbsDiff	0	$8,6$	$-4,6$	$-1,3$	$-0,4$
	RelDiff	0	∞	$-53,3\%$	$-31,8\%$	$-15,4\%$

Tabelle 6.20: Absolute und relative Differenz der Routenflexibilität und Produktionsrate in Evolutionsschritten der Fallstudie für CPPSe

Grundsätzlich lassen sich die Änderungen der Evolutionsszenarien an den nichtfunktionalen Eigenschaften ablesen. In den Evolutionsschritten $\Delta(\emptyset, m_{A1})$ und $\Delta(m_{A1}, \tilde{m}_{A2})$ werden neue Werkstücke eingeführt. Dies lässt sich an der neuen Produktionsrate von 8,6 mit einer unendlichen Steigerung und auch an einer veränderten Routineflexibilität ablesen. Das Experiment zeigt, dass bei einer unendlichen Erhöhung der relativen Differenz oft auf neue Ausprägungen geschlossen werden kann.

Die Einführung des Stempels mit dem Evolutionsschritt $\Delta(m_{A2}, \tilde{m}_{A3})$ hat eine verschlechterte Produktionsrate zur Folge, da Werkstücke langsamer produziert werden. An dieser Stelle lässt sich auch erkennen, warum nichtfunktionale Eigenschaften zur Bewertung nicht immer ausreichen, sondern weitere semantische Kontexte in Evolutionsschritten hilfreich sind. Denn die Einführung des Stempels ist nur sehr bedingt an den Planungs- und Fähigkeitsgrößen oder dem Modellverhalten abzulesen. Allerdings sind diese nicht Schwerpunkt der Modelle und auch nicht quantifiziert, weshalb an dieser Stelle zusätzliche semantische Kontextinformationen des Nutzers oder weitere Kontextquellen sinnvoll bzw. notwendig sind.

Wie bereits bei der Koevolution identifiziert, führt die Optimierung des Kranverhaltens in den betrachteten Eigenschaften zu einer Verschlechterung, weshalb dieser Evolutionsschritt nicht empfehlenswert ist. Die Erweiterung der Topologie durch das Laufband ist durch die erhöhte Routenflexibilität und Produktionsdauer zu erkennen.

Um die **Reaktion** der Evolutionsschritte zu bewerten, wurden die Verständlichkeit und Vollständigkeit der Operationen im Editierskript untersucht und es wurde dabei auch die Korrektheit der Anwendung des Editierskripts getestet.

Die Analyse der Operationen zeigt Tabelle 6.21. In den ersten Spalten ist der Evolutionsschritt und zur Komplexitätsabschätzung die Modellelemente des evolvierten Modells aufgeführt. Die weiteren Spalten geben die Anzahl der Anwendungen der Operationen im Editierskript an, wobei die letzte Spalte zusätzlich die Anzahl der primitiven Änderungen der Differenz angibt.

Insgesamt kann durch das Liften in Operationen eine Kompressionsrate von -82.79 % zwischen den Operationen und primitiven Änderungen erreicht werden. Zum Beispiel werden 152 primitive Änderungen des Evolutionsschrittes $\Delta(m_{A4}, \tilde{m}_{A5})$ in 23 Operationen ausgedrückt.

Evolutionsschritt	Modellelemente (Transition / Stelle)	Anwendungen der Operation											primitive Änderungen
		A	B	C	D	E	F	G	H	I	J	K	
$\Delta(\emptyset, m_{A1})$	5 / 4	4	1	0	0	0	13	5	4	0	0	0	162
$\Delta(m_{A1}, \tilde{m}_{A2})$	7 / 5	0	2	0	0	0	9	5	4	0	0	0	94
$\Delta(m_{A2}, \tilde{m}_{A3})$	16 / 13	8	0	0	0	1	24	8	9	0	0	3	310
$\Delta(m_{A3}, \tilde{m}_{A4})$	16 / 13	0	0	0	0	0	0	0	0	0	0	4	20
$\Delta(m_{A4}, \tilde{m}_{A5})$	19 / 16	4	0	0	0	0	3	8	8	0	0	0	152

Tabelle 6.21: Anwendungen der Operationen für die CPPS-Fallstudie

Die Operationsmenge ist bezüglich der Fallstudie dahingehend vollständig, als das alle primitiven Änderungen Teil von exakt einer Operationsanwendung sind. Diesbezüglich ist allerdings festzuhalten, dass mit der in dieser Arbeit definierten Operationsmenge nicht jede nach dem Meta-Modell valide Änderung erfasst wird. Die verwendeten Operationen wurden für eine gute Kompressionsrate und semantische Verständlichkeit auf die möglichen Modelle des Generierungsverfahren abgestimmt. Eine vollständige Abdeckung würde eine Menge von Operationen mit minimalen Konvergenzkriterien erreichen (vergleiche [KKT11]), was allerdings zu einer schlechten Kompression führen würde.

Bezüglich der einzelnen Operationen sind folgende Feststellungen für die Fallstudie abzuleiten. Die Struktur der Materialflussnetze ist hauptsächlich durch eine Anwendung der Operation A gekennzeichnet, da Erweiterungen am Ende des Produktionsprozesses vorgenommen werden. Ein Zusammenführen von Produktionspfaden kann durch Operation D-F abgedeckt werden. Neue Signalelemente (Operation F) treten hauptsächlich mit einer Strukturveränderung auf, was darauf schließen lässt, dass ein Zusammenführen dieser Regeln zu einer noch besseren Kompression führen sollte. Gleiches gilt für neue Zeitverhalten bestimmter Werkstücktypen (Operationen G und H).

Wie beim Grund des Evolutionsschrittes bereits festgestellt, treten neue zeitliche Bedingungen (Operation K) beim Warten auf andere Werkstücke und der Optimierung in Ausbaustufe A4 auf. Veränderungen des Ort- und Typkontextes sind in der Fallstudie nicht beobachtet worden. Dies resultiert aus der hohen Ähnlichkeit der Modellelemente zwischen den im Experiment untersuchten Versionen. Dies würde sich bei deutlicher unterscheidbaren Modellen oder bei einer weniger strengen Zuordnung ändern. Beispielsweise lässt sich bei einem reduzierten Schwellwert der Ähnlichkeit Θ_{lokal} im Zuordnungsverfahren beobachten, dass im letzten Evolutionsschritt Kransignale vor dem Laufband einer Lichtbarriere auf dem Laufband zugeordnet werden und dadurch Anwendungen der Operationen I und K entstehen.

Um die Korrektheit der Wirkung zu zeigen, wurde folgende drei Überprüfungen mit dem Editierskript und Modellkontext durchgeführt. Dabei sind die Überprüfungen anhand von Ausbaustufe A4 und A5 notiert, gelten aber für alle Evolutionsschritte:

1. Die Anwendung eines Evolutionsschrittes auf das Ursprungsmodell führt zum Folgemodell: $m_{A4} + \Delta(m_{A4}, \tilde{m}_{A5}) = \tilde{m}_{A5}$
2. Der invertierte Evolutionsschritt[37] führt bei Anwendung auf das Folgemodell zur Wiederherstellung des Ursprungsmodell: $m_{A5} + \Delta(m_{A5}, \tilde{m}_{A4}) = \tilde{m}_{A4}$
3. Eine mehrfache Anwendung von Evolutionsschritten führt zum gleichen Ergebnis wie einfache Anwendungen: $m_{A3} + \Delta(m_{A3}, \tilde{m}_{A4}) + \Delta(m_{A4}, \tilde{m}_{A5}) = m_{A4} + \Delta(m_{A4}, \tilde{m}_{A5}) = m_{A5}$.

Im Allgemeinen lassen die Experimentergebnisse folgende Schlussfolgerungen zu:

- Die Abweichungen der Klasse *Einfügung* zeigten die häufigste und beste Indikation für den Grund einer Evolution. Auch der *Wegfall* von Ereignissen ist als guter Indikator anzusehen, wobei Systeme durch Evolution in der Regel komplexer werden [Leh96] und diese Klasse durch Fehlverhalten häufiger auftritt. Die weiteren Klassen wurden in der Fallstudie als system- und konfigurationsabhängiger eingestuft und sind deshalb schwerer zu bewerten.
- Grundsätzlich können die Ergebnisse des Evolutionsschrittes durch die gewählten Eigenschaften beschrieben werden. Es hat sich jedoch gezeigt, dass weitere semantische Eigenschaften für eine umfängliche Bewertung zusätzlich notwendig sind. Deshalb scheint der halbautomatisierte Ansatz in dieser Arbeit sinnvoll. Für weitere Experimente wurden deshalb zusätzlich auch einige manuelle Kontextinformation eingeführt, die die Größe der Erweiterung des Funktionsumfangs beschreiben.
- Die Operationsmengen bilden alle in der Fallstudie vorgenommenen Änderungen ab. Das semantische Liften in Operationen ermöglicht eine erhebliche Kompression und damit bessere Verständlichkeit dieser Änderungen. Bei der Auswertung einzelner Operationen hat sich gezeigt, dass eine weitere Kompression möglich ist indem z.B. Operationen zusammengefasst werden.
- Die Anwendung hat sich bei der gewählten Überprüfung auf Ursprungsmodelle als korrekt erwiesen.

[37] Dazu wurden die Graphoperationen des Hinzufügens und Entfernens invertiert und der Modellkontext aus dem Folgemodell verwendet.

Vergleichsprozess über einen verteilten Marktplatz

In einem weiteren Experiment wird die Evolutionsunterstützung durch Vorschläge von Änderungen evaluiert.

Experimentaufbau: Die Evolutionsschritte des vorigen Experiments werden in drei wissenstragenden Komponenten genutzt, damit der Austausch evaluiert werden kann. Dafür wird die zweite und dritte PPU-Komponente verwendet.[38] Diese wissenstragenden Komponenten laufen auf ihren eigenen Rechensystemen[39] und sind über ein Netzwerk mit den anderen wissenstragenden Komponenten verbunden. Alle wissenstragenden Komponenten besitzen dafür eine Blockchain-Komponente. Die wissenstragenden Komponenten sind hierbei im Genesisblock der Blockchain festgelegt, der vor dem Experiment jeder Blockchain-Komponente bekannt gemacht wurde.

Als Konfiguration der Evolutionsunterstützungskomponente wird die in Abschnitt 3.6.1 definierte Bewertung verwendet. Dabei verwendet die zweite PPU-Komponente als Präferenz das Einfügen zusätzlicher Kapazitäten und Routen. Die dritte PPU-Komponente präferiert dagegen die Erweiterung ihres Funktionsumfang. Es wird im Experiment immer der Evolutionsschritt mit der besten Bewertung zur Anwendung ausgewählt.

Machbarkeitsstudie: Um die Unterstützung durch Vorschläge aus Versionen zu evaluieren, müssen Vorschläge abgeleitet und bewertet werden. Anschließend sind diese im Evolutionsprozess zu dokumentieren.

Zunächst wird die **Anwendung** des zuvor betrachteten Evolutionsschritt $\Delta(m_{A4}, \tilde{m}_{A5})$ der ersten PPU-Komponente auf die Grundstufe B1 der zweiten PPU-Komponente betrachtet. Dazu ist die Wirkung mit dem Editierskript $\delta(m_{A4}, \tilde{m}_{A5})$ und dem Modellkontext $\delta(m_{A4}, \tilde{m}_{A5})_{kontext}$ auf das neue Modell $\hat{m}_{B1}$ anzuwenden. Der Modellkontext ist durch den Kontext der Stelle s_{14} gegeben, die der einzige unbelegte Parameter des zugehörigen Editierskripts ist.

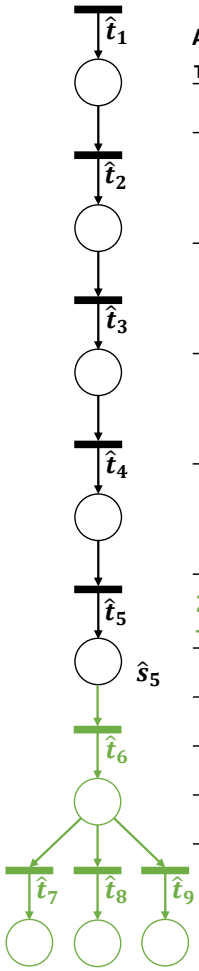
Die nach Abschnitt 3.5.2 bestimmte Modellzuordnung $z(\delta(m_{A4}, \tilde{m}_{A5})_{kontext}, \hat{m}_{B1})$ enthält für s_{14} die Zuordnung $(s_{14} \rightarrow \hat{s}_5)$. Das Editierskript wird somit ausgeführt indem die ursprüngliche Stelle s_{14} durch $\hat{s}_5$ in $\hat{m}$ ersetzt wird. Die daraus resultierende Handlungsalternative $\hat{m}_{B1}^{\Delta(m_{A4}, \tilde{m}_{A5})}$ zeigt Abbildung 6.16. Diese zeigt, dass die Transitionen des Laufbands an das Ende der Transitionsfolge von Grundstufe B1 angefügt wurden.

Für die Handlungsalternative ergibt sich hier die folgende **Bewertung**:

- Der **Grund** des Evolutionsschrittes war, dass eine Abweichung der Klasse *Einfügung* auftrat. Diese resultierte aus der späteren zusätzlichen Transition $\hat{t}_{15}$ der Lichtschranke. Der Nutzer hat entsprechend seiner Präferenz die maximale Bewertung für die Klasse *Einfügung* und neue Ereignisse der Werkstückerkennung festgesetzt.[40] Die Anwendungsbewertung des Grunds lässt sich somit wie folgt berechnen:[41]

$$f_{Grund} = \frac{5 \cdot 1 + 1 + 1}{5 + 2} = 1 \tag{6.9}$$

[38] Das Experiment wurde nur auf Basis der Modelle, also ohne Verbindung zur eigentlichen Laboranlage durchgeführt.
[39] Virtuelle Maschine, 2 Kern CPU, 8 GB RAM
[40] Die spezifizierten Regeln sind $bewertung^{abw}_{Einfügen} = 1$ und $bewertung^{e}_{(typ,\ Sensor-Werkstück-Erkennung)} = bewertung^{\tilde{e}}_{(typ,\ \square)} = 1$
[41] In der Evaluation wurde der Multiplikator $k_{Klasse} = 5$ als optimaler Wert für die Fallstudie identifiziert.



Ausbaustufe A1

Transition	Signal	Typ	Ort	Zeit
$\hat{t}_1$	S.WorkpieceReady	Sensor-Werkstück-Erkennung	Magazin	Plastik 2479 ms
$\hat{t}_2$	S.Sucked	Aktor-Werkstück-Halten	Crane	Plastik 119 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnMagazin	Sensor-XPosition	Magazin	
$\hat{t}_3$	S.Sucked	Aktor-Werkstück-Halten	Crane	Plastik 122 ms
	S.Up	Sensor-YPosition	Crane	
	S.CraneOnMagazin	Sensor-XPosition	Magazin	
$\hat{t}_4$	S.Sucked	Aktor-Werkstück-Halten	Crane	Plastik 300 ms
	S.Up	Sensor-YPosition	Crane	
	S. CraneOnConveyor	Sensor-XPosition	Conveyor	
$\hat{t}_5$	S.Sucked	Aktor-Werkstück-Halten	Crane	Plastik 120 ms
	S.Down	Sensor-YPosition	Crane	
	S.CraneOnConveyor	Sensor-YPosition	Conveyor	

Zusätzliche Modelltransition der Handlungsalternative

Transition	Signal	Typ	Ort	Zeit
$\hat{t}_6$	S.Conveyor-Lightbarrier	Sensor-Werkstück-Erkennung	Conveyor	Plastik 640 ms
$\hat{t}_7$	S.SorterPusher1-MovedOut	Sensor-Werkstück-Bewegen	Conveyor	Plastik 121 ms
$\hat{t}_8$	S.PositionRamp3	Sensor-Werkstück-Erkennung	Conveyor	Plastik 419 ms
$\hat{t}_9$	S.SorterPusher2-MovedOut	Sensor-Werkstück-Bewegen	Conveyor	Plastik 120 ms

Abbildung 6.16: Handlungsalternative der Grundstufe B1 nach Anwendung des Evolutionsschrittes $\Delta(m_{A4}, \tilde{m}_{A5})$

- Die Zuordnung des Eingangsparameters $\tilde{s}_{14} \leftrightarrow \hat{s}_{14}$ erfolgt mit einer Ähnlichkeit von 0,5, da die Transitionen vor der Stelle mit einer Wahrscheinlichkeit 1 zugeordnet wird und keine nachfolgende Transition in A4 existiert. Somit ist die Bewertung der **Reaktion** $f_{Reaktion} = 0,5$.
- Bezüglich des **Ergebnisses** kann die Berechnung, für die im vorherigen Abschnitt gezeigte Anwendung des Evolutionsschrittes, verwendet werden. Dadurch ergibt sich die absolute und relative Differenz der Produktionsrate von $-3,11$ und $-36,36\%$ und der Routenflexibilität von $0,83$ und ∞. Durch die Anwendung der Verfahrensvorschrift 6 ist die Anwendungsbewertung des Ergebnisses dann $f_{Ergebnis} = 0,6085$.
- Daraus ergibt sich unter einer geringeren Bewertung des Grunds folgende endgültige **Bewertung** des Evolutionsschrittes:[42]

$$f_{Bewertung}(\Delta(m, \tilde{m}), \hat{m}) = \frac{0,5 \cdot 1 + 1 \cdot 0,5 + 1 \cdot 0,6085}{0,5 + 1 + 1} = 0,6434$$

[42] Es wurde $\omega_{Grund} = 0,5$, $\omega_{Reaktion} = 1$ und $\omega_{Ergebnis} = 1$ angenommen.

Im **Evolutionsprozess** ist für die Nachfolgerelation zunächst die Ausgangslage der Fallstudie zu berücksichtigen:

1. $\phi \succ_{ZR} \Delta(\phi, m_{A1})$
2. $\phi \succ_{ZR} \Delta(\phi, m_{B1})$
3. $\phi \succ_{ZR} \Delta(\phi, m_{C1})$
4. $\Delta(\phi, m_{A1}) \succ_{ZR} \Delta(m_{A1}, m_{A2})$
5. $\Delta(m_{A1}, m_{A2}) \succ_{ZR} \Delta(m_{A2}, m_{A3})$
6. $\Delta(m_{A2}, m_{A3}) \succ_{ZR} \Delta(m_{A3}, m_{A4})$
7. $\Delta(m_{A3}, m_{A4}) \succ_{ZR} \Delta(m_{A4}, m_{A5})$

Die Ausgangslage beschreibt die mehrfach evolvierte erste PPU-Komponente und die zwei weiteren, noch nicht evolvierten PPU-Komponenten. Eine Vererbung liegt in der Ausgangslage nicht vor. Wenn allerdings der in der Machbarkeitsstudie bewertete Evolutionsschritt implementiert wird, entsteht folgende Vererbungsrelation sowie Nachfolgerelation:

1. $\Delta(m_{A4}, m_{A5}) \succ_{VV} \Delta(m_{B1}, m_{B2})$
2. $\Delta(\phi, m_{B1}) \succ_{ZR} \Delta(m_{B1}, m_{B2})$

Der nach einer Implementierung neu gebildete Evolutionsschritt $\Delta(m_{B1}, m_{B2})$ beschreibt das Hinzufügen des Laufband in der zweiten PPU-Komponente und wurde aufgrund des Vorschlags aus dem Evolutionsschritt $\Delta(m_{A4}, m_{A5})$ der ersten PPU-Komponente realisiert.

Diskussion der Ergebnisse

Im Experiment wurden die vorhandenen Evolutionsschritte über Evolutionsschrittkomponenten angeboten. Diese konnten von den verschiedenen wissenstragenden Systemen genutzt werden. Dabei wurden die jeweiligen Evolutionsschritte in die Blockchain eingefügt und über die Evolutionsunterstützungskomponente bewertet. Abbildung 6.17 zeigt den dadurch entstandenen Evolutionsprozess. Dabei wurden - mit den definierten Präferenzen der zweiten und dritten PPU-Komponente - jeweils der am besten bewertete Evolutionsschritte ausgewählt.

Wie die Abbildung zeigt, wird für die Grundstufe B1 der Evolutionsschritt $\Delta(m_{A1}, \tilde{m}_{A2})$ ausgewählt. Denn die Modelle B1 und A1 sind identisch. Entsprechend wird mit metallischen Werkstücken eine zusätzliche Werkstückausprägung hinzugefügt. Die Handlungsalternative prognostiziert eine Steigerung der Routenflexibilität und keine Beeinflussung der Produktionsrate (vergleiche Tabelle 6.20). Da eine Implementierung dieses Vorschlages angenommen wird, wird eine räumlich-zeitlichen Relation und eine Vererbungsrelation erstellt (gestrichelte Linie in Abbildung 6.17).

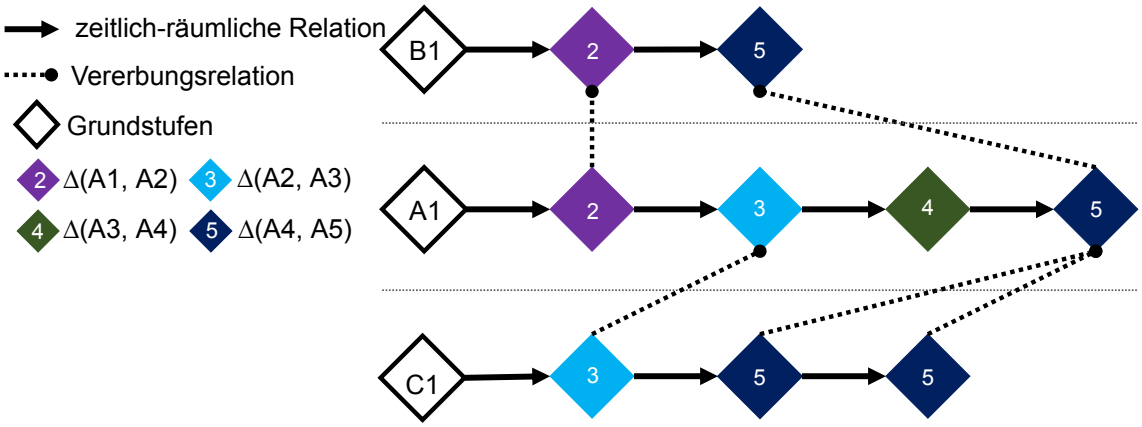


Abbildung 6.17: Evolutionsprozess der drei PPU-Komponenten

Anschließend wird durch diese PPU-Komponente der Evolutionsschritt $\Delta(m_{A4}, \tilde{m}_{A5})$ präferiert, da er insbesondere ein besseres Ergebnis als eine erneute Anwendung von $\Delta(m_{A1}, \tilde{m}_{A2})$ verspricht. Das resultierende Materialflussmodell enthält, wie in Abbildung 6.16 gezeigt, dadurch ein Laufband ohne dass metallische Werkstücken produziert werden.

Bei der Grundstufe C1 mit nur metallischen Werkstücken wird hingegen zunächst ein Stempel hinzugefügt. Dies resultiert aus der unterschiedlichen Präferenz der zweiten und dritten PPU-Komponente. Das Hinzufügen des Stempels hat eine Funktionserweiterung als Ergebnis[43]. Anschließend wird der Evolutionsschritt $\Delta(m_{A4}, \tilde{m}_{A5})$ angewendet.

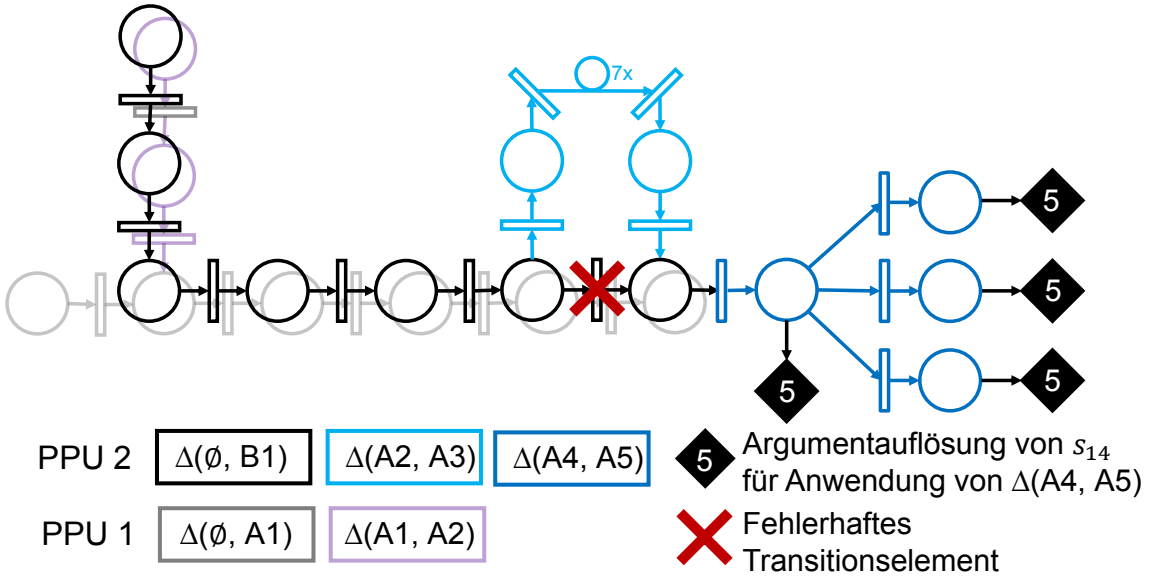


Abbildung 6.18: Handlungsalternative der dritten PPU-Komponente

Das resultierende Materialflussmodell nach dieser Anwendung ist in Abbildung 6.18 dargestellt. Das initiale Modell der Grundstufe C1 ist in schwarz dargestellt. Durch den Evolutionsschritt $\Delta(m_{A2}, \tilde{m}_{A3})$ wird der Stempel hinzugefügt. An dieser Stelle besitzt das Modell durch die Modelltransformation eine Transition, die durch eine Beobachtung des operativen Systems nicht entstehen würde. Denn die letzte Transition des ursprünglichen Modells beschreibt das Abstellen des Werkstückes an der Ausgangsrampe, wenn der Kran vom Magazin kommt (vergleiche transparente Darstellung der ersten PPU-Komponente in Abbildung 6.18). Dieses Abstellen würde bei einer operativen Implementierung nicht beobachtet werden, da in der dritten PPU-Komponente keine Plastikwerkstücke existieren, die direkt vom Magazin zur Ausgangsrampe transportiert werden. Obwohl der Evolutionsschritt die Konfiguration für metallische Werkstücke löscht, gibt es bei dieser Transformation keine Löschung der Transition durch die Operationen des Evolutionsschrittes. Hier müsste eine zusätzliche Validierung des transformierten Modells verwendet werden, die die Transition ohne Konfiguration löscht. Dies könnte, wie im Ausblick in Abschnitt 7.3 erläutert, Teil weiterer Forschungsarbeiten sein.

Abbildung 6.18 zeigt weiterhin auch die Anwendung des Evolutionsschrittes $\Delta(m_{A4}, \tilde{m}_{A5})$. Dieser wird zweimalig angewendet, da er nach einer ersten Anwendung weiterhin die höchste Bewertung besitzt. Allerdings ist die Argumentauflösung verändert indem nun die Transition für das Ereignis am Ende des ersten Laufbands zugeordnet wird. Dies entspräche auch in der operativen PPU-Komponente einem Hinzufügen eines zweiten Laufbands, welches direkt an das erste Laufband anschließen würde. Dabei besitzen auch die Stellen der durch die

[43] Dies wird nur durch den manuell hinzugefügten Kontext der Funktionserweiterung erkannt

Pneumatikzylinder bedienten Ausgangsrampen eine hohe Ähnlichkeit zum Argument des Modellkontextes des Evolutionsschrittes. Entsprechend können Laufbänder auch als Alternativen zu den weiteren zwei Ausgangsrampen verwendet werden. Diese mögliche Argumentauflösung für die zweite Anwendung des Evolutionsschrittes ist in Abbildung 6.18 visualisiert. Es zeigt, dass ein modellbasierter Vorschlag operativ sinnvolle Empfehlungen gibt.

Bezüglich des **Marktplatzes** wurde zusätzlich die *Skalierung* der prototypischen Implementierung untersucht. Unter dem gegebenen Experimentaufbau wurden dabei unterschiedliche Mengen von wissenstragenden Komponenten auf jedem der drei Rechensysteme der PPU-Komponenten gestartet. Dabei wurden durch die wissenstragenden Komponenten immer wieder Evolutionsschritte ausgewählt und angewendet. Aus den resultierenden Handlungsalternativen wurden Evolutionsschritte abgeleitet und in die Blockchain eingetragen.

Abbildung 6.19 zeigt den Verlauf für unterschiedliche Mengen von wissenstragenden Komponenten. Dabei zeigt sich, dass die Netzwerklast grundsätzlich linear mit der Anzahl von Evolutionsschritten ansteigt. Bei etwa 900 Evolutionsschritten wird die Last auf das Netzwerk größer, sodass die Rechenkapazität für die Erzeugung von neuen Evolutionsschritten abnimmt. Eine Analyse zeigt, dass für die Erhöhung dieser Kapazitätsgrenze vor allem die Implementierung der Anwendung des Editierskripts verbessert werden müsste, da die Modelle in ihrer Komplexität stetig wachsen und dabei die Systeme überlasten.

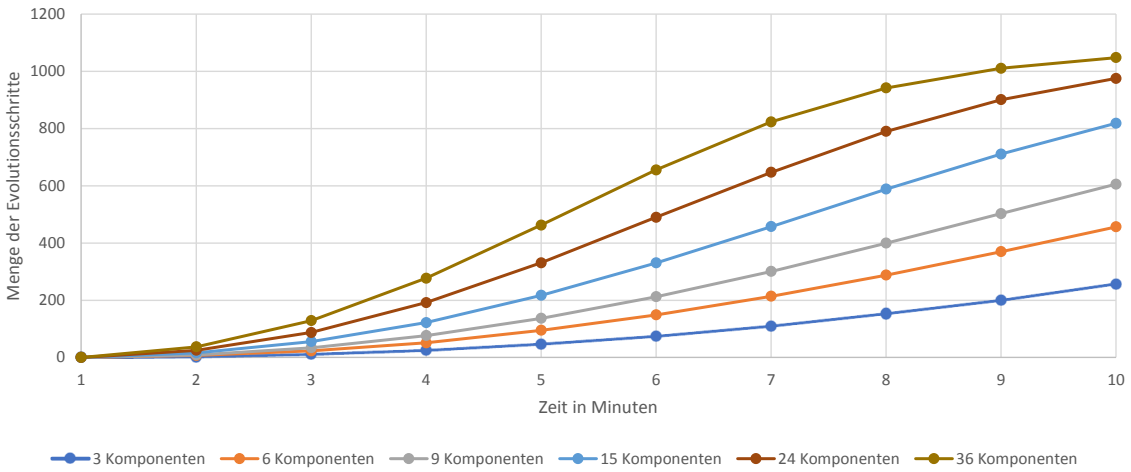


Abbildung 6.19: Skalierung der wissenstragenden Komponenten bezüglich der Menge von Evolutionsschritten

Aus dem gesamten Experiment lassen sich folgende allgemeine Schlussfolgerungen ableiten:

- Durch die Modelltransformation konnten in der Fallstudie operativ sinnvolle Vorschläge erzeugt werden.
- Eine Anwendung auf ähnliche Modelle kann vorgenommen werden. Diese können einen Mehrwert für den Benutzer bieten und bilden das zu implementierende CPS durch modellbasierte Handlungsalternativen grundsätzlich gut ab. Der entstehende Evolutionsprozess kann dokumentiert werden.
- Es zeigt sich, dass die Handlungsalternativen Modellfehler beinhalten können. Diese werden allerdings durch die spätere Implementierung revidiert und können durch unterschiedliche Modellkontexte eines Evolutionsschrittes bei einer größeren Auswahl von vererbten Evolutionsschritten reduziert werden.

- Eine logische Validierung der Implementierbarkeit der Handlungsalternativen ist mit den in dieser Arbeit vorgestellten Methodiken noch nicht möglich. Dennoch zeigte die Fallstudie nur wenige dieser nicht implementierbaren Handlungsalternativen und es könnten Verfahren für eine nachträgliche Validierung integriert werden.
- Durch die vorgenommene Bewertung können Eigenschaften vorhergesagt werden, wodurch präferierte Evolutionsschritte gefunden werden können. Für eine differenzierte Bewertung müssten insbesondere weitere semantische Kontexte ergänzt werden.
- Die Untersuchung des Marktplatzes zeigte, dass für die Fallstudie auch mit beschränkten Rechenressourcen eine lineare Skalierung von bis zu dreißig Evolutionsschritten pro wissenstragender Komponente erreicht werden kann. Als Engpass wurde dabei nicht die Systemarchitektur an sich, sondern die prototypisch implementierte Modelltransformation identifiziert.

Zusammenfassende Bewertung des vierten Evaluationsziels

In den beiden Experimenten wurde gezeigt, dass Evolutionsschritte aus den Modellversionen der Fallstudie für CPPSe abgeleitet werden können. Diese bilden unterschiedliche Gründe von Evolution, eine anwendbare, modellbasierte Änderung und die Eigenschaften des Ergebnisses ab. Auch wenn eine semantische Validierung auf das nach außen beobachtbare Verhalten beschränkt ist, konnte eine sinnvolle Evolutionsunterstützung erreicht werden. Die abgeleiteten Vorschläge sind im Sinne des operativen CPS als valide einzustufen, auch wenn sie nicht vollständig fehlerfrei sind und auf der Modellebene verbleiben. Für eine umfassende Validierung hat sich somit die Einbeziehung des Nutzers weiterhin als essentiell herausgestellt.

Mit der wissenstragenden Komponente und ihrer Verknüpfung in einem Marktplatz wurde eine im Rahmen der Fallstudie gut skalierende Systemarchitektur geschaffen. Diese ermöglicht es, Evolutionsschritte zu bilden, auszutauschen und anzuwenden. Der dadurch operativ fortgeschriebene Evolutionsprozess kann vollständig dokumentiert werden.

6.3.5 Ähnlichkeitsmaß für Varianten

Im letzten vorgenommenen Experiment werden als zusätzliches Ziel Evolutionsschritte genutzt, um Ähnlichkeiten von Varianten zu definieren.Für dieses Experiment werden zunächst aus zwei CBCPS-Modellen Evolutionsschritte abgeleitet. Diese werden dann genutzt, um ein modellbasiertes Ähnlichkeitsmaß abzuleiten. Ein vergleichbares operationsbasiertes Ähnlichkeitsmaß wird in unterschiedlichen Forschungsbereichen zur Bestimmung von Ähnlichkeiten eingesetzt [CMZ09]. Mit Evolutionsschritten kann ein solches Ähnlichkeitsmaß über die drei Hauptdimensionen von Änderungen erstellt werden und es kann zusätzlich der Unterschied durch die Reaktion modellbasiert beschrieben werden.

Experimentaufbau: Die Modelle werden entsprechend des Experimentaufbaus für Modellartefakte von CBCPSen erzeugt. Für jede Methode werden dann Methodenverhaltensmodellen generiert, die einer Evolutionsschrittkomponente über Dienstaufrufe mitgeteilt werden. Der Aufbau zur Bildung von Evolutionsschritten gleicht weiterhin den Experimenten für Versionen der PPU-Fallstudie.

Machbarkeitsstudie: Da in diesem Experiment Varianten betrachtet werden, werden als Beispiel die jeweiligen Methodenverhaltensmodelle für zwei unterschiedliche, aber ähnliche Methoden betrachtet. Dazu wurde das Methodenverhaltensmodell von der Methode *KassenApplikation.barcodescanned* bereits in Abbildung 6.10 gezeigt. Das Modell zeigt ea_1, ea_2, ea_3 als einzige Folge von *ExterneAktion*-Elementen. Die in der Ausbaustufe hinzukommende Methode *KassenApplikation.addLastProduct*, welche über eine Taste der Benutzeroberfläche den letzten Artikel wiederholt eingibt, wird dazu verglichen.

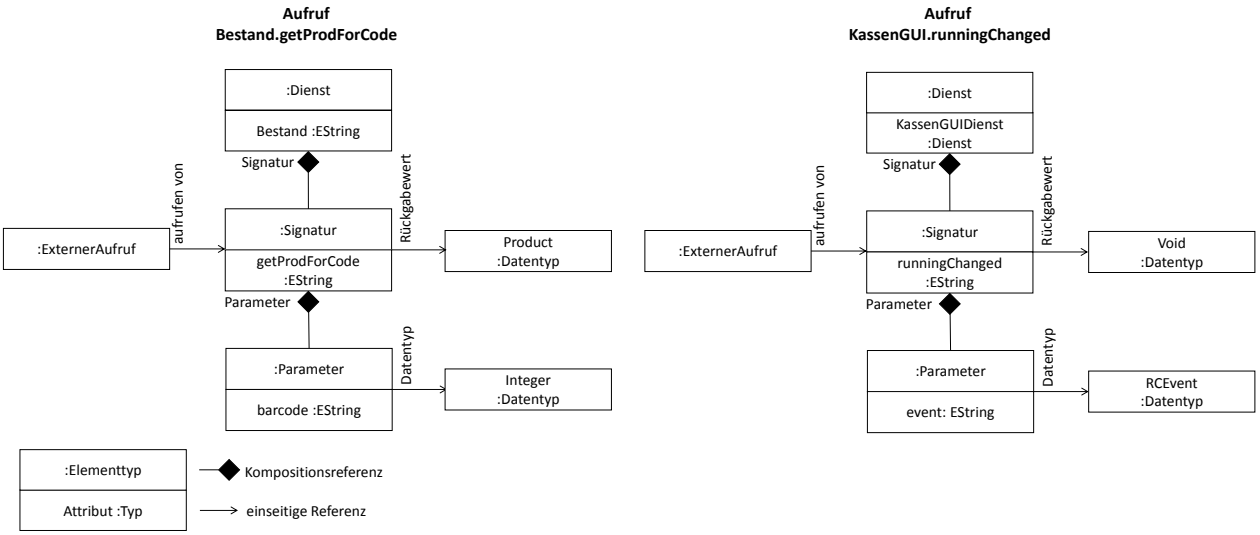


Abbildung 6.20: Elemente des abstrakten Syntaxgraphen des Repositorymodells von zwei externen Aktionen

Für eine **Zuordnung** ist die Ähnlichkeit der Kontexte der *ExterneAktion*-Elemente vorzunehmen. Abbildung 6.20 zeigt die zwei Ausschnitte der Methoden im abstrakten Syntaxgraph des Repositorymodells. Auf der linken Seite werden hier die Modellelemente gezeigt, zu denen der erste externe Aufruf ea_1 von der Methode *KA.barcodesanned* eine Assoziation besitzt, und auf der rechten Seite die Elemente des ersten Aufrufs $\tilde{ea}_1$ der Methode *KA.addLastProduct*.

Für diese Aktionen wird nachfolgend die Kontextbestimmung gezeigt.

- **Zeitkontext:** Bezüglich des Zeitkontextes ist eine iterative Bestimmung nicht notwendig, da keine weiteren verschachtelten Aufrufe vorliegen. Der Zeitkontext ergibt sich aus dem Bedarf der *InterneAktion*-Elemente. Diese sind $\mu = 79$ und $\tilde{\mu} = 23,9$. Daraus ergibt sich der Standardfehler der Mittelwertdifferenz $92,17$ und somit die Ähnlichkeit des Zeitkontextes von $0,783$.[44]
- **Ortskontext:** Bei die Anwendung der Jaro-Distanz auf den Ortkontext ergeben sich bei den Namen $d = BestandsDienst$ und $\tilde{d} = KassenGUIControllerDienst$ die beiden Zeichenreihen $\begin{matrix} assennteDist \\ estansDienst \end{matrix}$.[45] Aus diesen lässt sich die Ähnlichkeit des Ortskontextes berechnen:[46]

$$ähnlichkeit^{ortskontext}_{ea_1 \to \tilde{ea}_1} = \frac{1}{3}\left(\frac{12}{25} + \frac{12}{14} + \frac{12 - \frac{8}{2}}{12}\right) = 0,67 \tag{6.10}$$

- **Typkontext:** Für den Vergleich der Typkontexte[47] muss insbesondere das Parameterpaar $(p_1, \tilde{p_1})$ mit seinen Namen und Datentypen verglichen werden. Es ergeben sich die Ähnlichkeit des Namens $barcode \to event = 0$ und des Datentyps $Integer \to RCEvent = 0,43$. Somit ist die Ähnlichkeit des Parameterpaars $ähnlichkeit_{p_1,\tilde{p_1}} = 0,25$.[48] Für den Rückhabetyp *Void*, der nicht vorhandenen Rückgabe, wird die Wahrscheinlichkeit bei Gleichheit 1 und ansonsten, wie in diesem Fall, 0 angenommen. Als Jaro-Ähnlichkeiten ergibt sich unter Gleichgewichtung der Ähnlichkeit somit:

$$ähnlichkeit^{Ortskontext}_{ea_1 \leftrightarrow \tilde{ea}_1} = \frac{0,43 \cdot 1 + 0,25 \cdot 1 + 0 \cdot 1}{1 + 1 + 1} = 0,23 \tag{6.11}$$

- Unter einer Gleichgewichtung der Kontexte ergibt sich somit die endgültige **Ähnlichkeit** von 0,56.

Tabelle 6.22 zeigt die Ähnlichkeiten aller *ExterneAktion*-Elemente (betrachtete Fall rot markiert). Daraus folgt nach Verfahrensvorschrift 3, dass die nahezu identischen *ExterneAktion*-Elemente $ea_2 \leftrightarrow \tilde{ea}_1$ und $ea_3 \leftrightarrow \tilde{ea}_2$ zugeordnet werden.

	$\tilde{ea}_1$	$\tilde{ea}_2$
ea_1	0,56	0,33
ea_2	1	0,9
ea_3	0,9	1

Tabelle 6.22: Ähnlichkeiten der Elemente vom Typ *ExterneAktion* der Methoden *KA.addLastProduct* und *KA.barcodeScanned*

[44] Weitere Werte: $n = \tilde{n} = 300$, $\sigma^2 = 17,64$, $\tilde{\sigma}^2 = 90,25$ und $\Theta_{minZeit} = 100$

[45] Die Distanz ist $\frac{max(|25|, |\tilde{14}|)}{2} - 1 = 11$

[46] Es ergibt sich $m = 12$, da es sich um jeweils 11 Zeichen handelt, und $t = 8$, da acht dieser Zeichen nicht direkt an der richtigen Position stehen und somit Vertauschungen erfordern.

[47] Der Typkontext ist für ea_1 durch das Namensattribut $s = getProductForCode$, das dazugehörigen Parameterobjekt p_1 mit seinem Namen $pn_1 = barcode$ sowie die Typen der Datentypen $tp_1 = Integer$ des Parameters und dem Rückgabewert $tr = Product$ gegeben.

[48] Die Lösung des Zuordnungsproblems ist bei nur einem Parameter nicht notwendig.

Die Zuordnung von Elementen des Typs *InterneAktion* wird danach entsprechend ihrer Abhängigkeit zu den *ExterneAktion*-Elementen vorgenommen. Dies wird in Abbildung 6.21 gezeigt. Dabei wurde die Nachfolgerelation $\succ_N$ genutzt. Diese sagt beispielsweise aus, dass das dritte Element ia_3 des Typs *InterneAktion* der ersten Folge genau nach ea_2 erfolgt. Diese Nachfolge wird entsprechend der Zuordnungen auf $\widetilde{ia}_2$ übertragen.

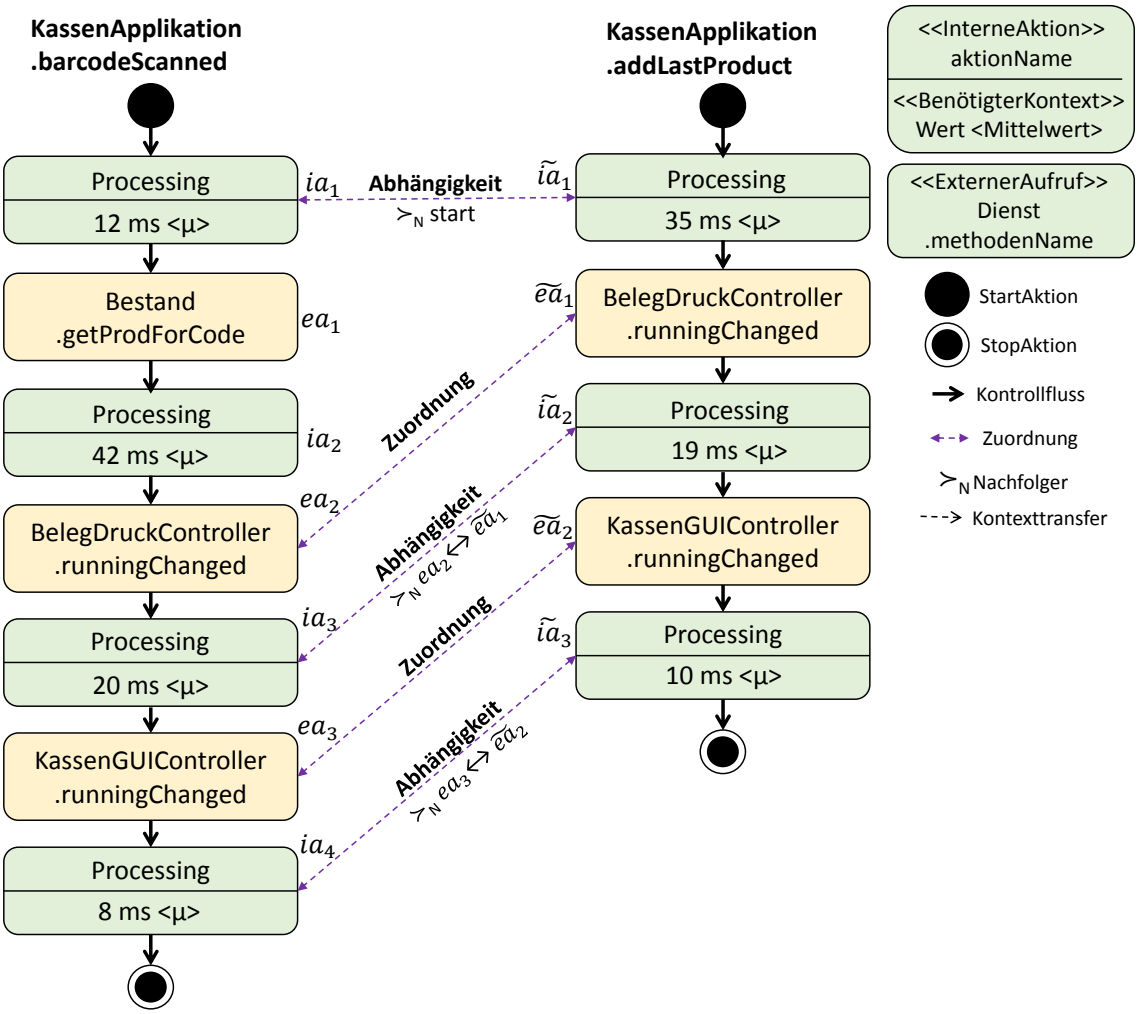


Abbildung 6.21: Zuordnung von *InterneAktion*-Elementen durch Abhängigkeiten der Nachfolgerelation

Beim Vergleich der Nachrichtensequenzen der Varianten treten als **Grund** zwei Abweichungen auf: Dies sind Abweichungen der Klasse *Wegfallen* und *Zeitaberration*. Denn das Ereignis des Methodenaufrufs von *Bestand.getProdForCode* wird in der Nachrichtensequenz der Variante nicht mehr beobachtet und durch einen leicht erhöhten Bedarf an Zeit kompensiert.

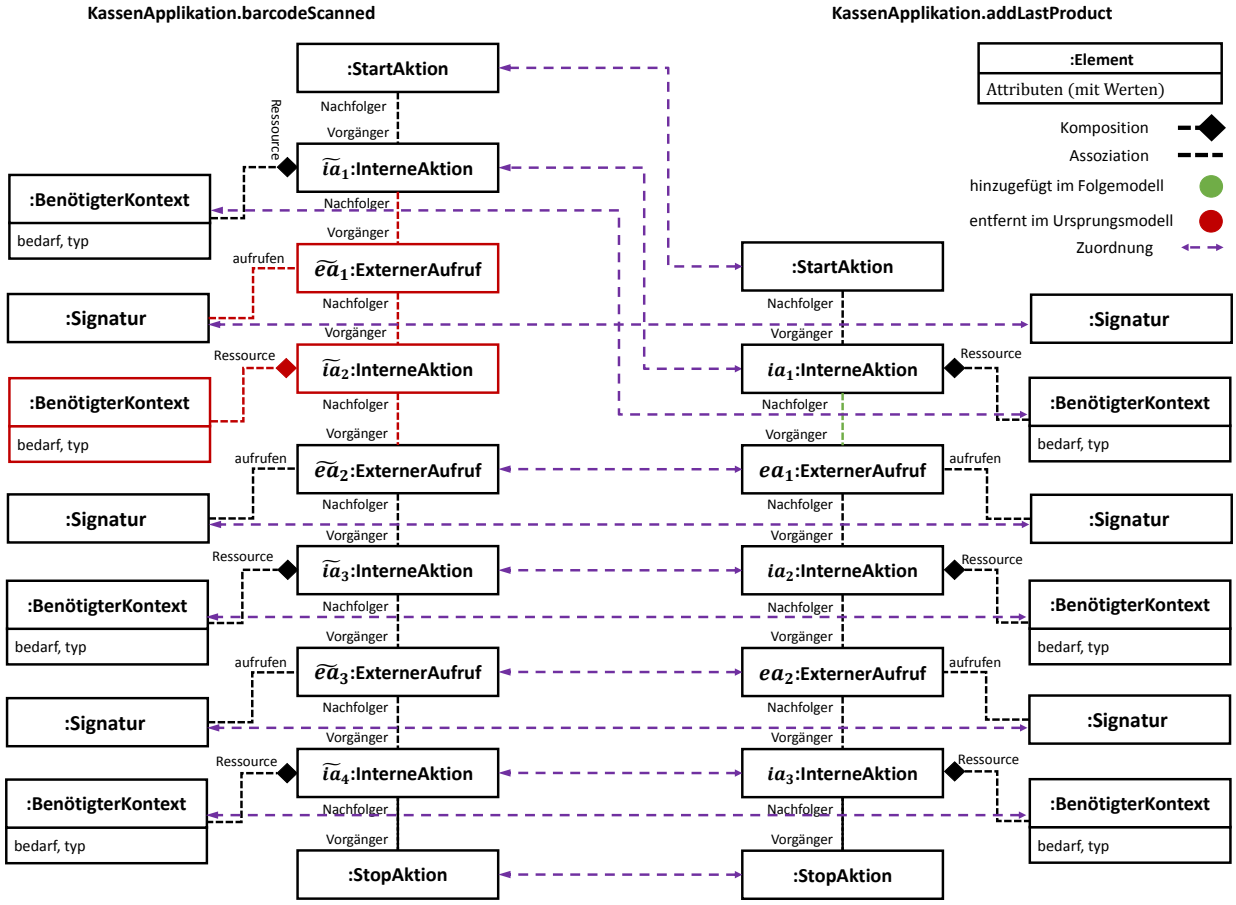


Abbildung 6.22: Ausschnitt des abstrakten Syntaxgraphen der Methoden *KA.barcodeScanned* und *KA.addLastProduct* mit Zuordnung und Markierung der hinzugefügten und entfernten Elemente

Zur Bestimmung der **Reaktion** unter dieser Zuordnung zeigt Abbildung 6.22 einen Ausschnitt des abstrakten Syntaxgraphen mit folgenden primitiven Änderungen:

1. $erzeugeReferenz(Vorgänger, ea_1, ia_1)$
2. $erzeugeReferenz(Nachfolger, ea_1, ia_1)$
3. $entferneElement(ExterneAktion)$
4. $entferneReferenz(Vorgänger, ExterneAktion, \tilde{ia}_1)$
5. $entferneReferenz(Nachfolger, \tilde{ia}_1, ExterneAktion)$
6. $entferneReferenz(aufrufen, ExterneAktion, Bestand.getProdForCode)$
7. $entferneElement(InterneAktion)$
8. $entferneReferenz(Vorgänger, InterneAktion, ExterneAktion)$
9. $entferneReferenz(Nachfolger, ExterneAktion, InterneAktion)$
10. $entferneReferenz(Vorgänger, \tilde{ia}_2, InterneAktion)$
11. $entferneReferenz(Nachfolger, InterneAktion, \tilde{ia}_2)$
12. $entferneElement(BenötigterKontext, (\mu, \sigma, min, max, n), zeitkontext")$
13. $entferneReferenz(Ressource, InternerAktion, BenötigterKontext)$
14. $erzeugeReferenz(Vorgänger, ea_1, ia_1)$
15. $modifiziereElement(ia_1, bedarf, 30ms)$

Zunächst existieren für das Hinzufügen der neuen Referenzen zwei primitive Änderung (1.+2.). Zusätzlich werden ein *ExterneAktion*-Element und ein *InterneAktion*-Element entfernt (3.+7.). Diese wurden auch von den vorherigen Elementen getrennt (4.+5.+8.-12.). Auch die Referenz zur vorhandenen Signatur des Repositorymodells wird entfernt (6.).[49] Als letzte primitive Änderungen wird das *BenötigteKontext*-Element entfernt (13.) und getrennt (14.). Zusätzlich wird das Attribut $bedarf$ des *InterneAktion*-Elements ia_1 verändert (15.).

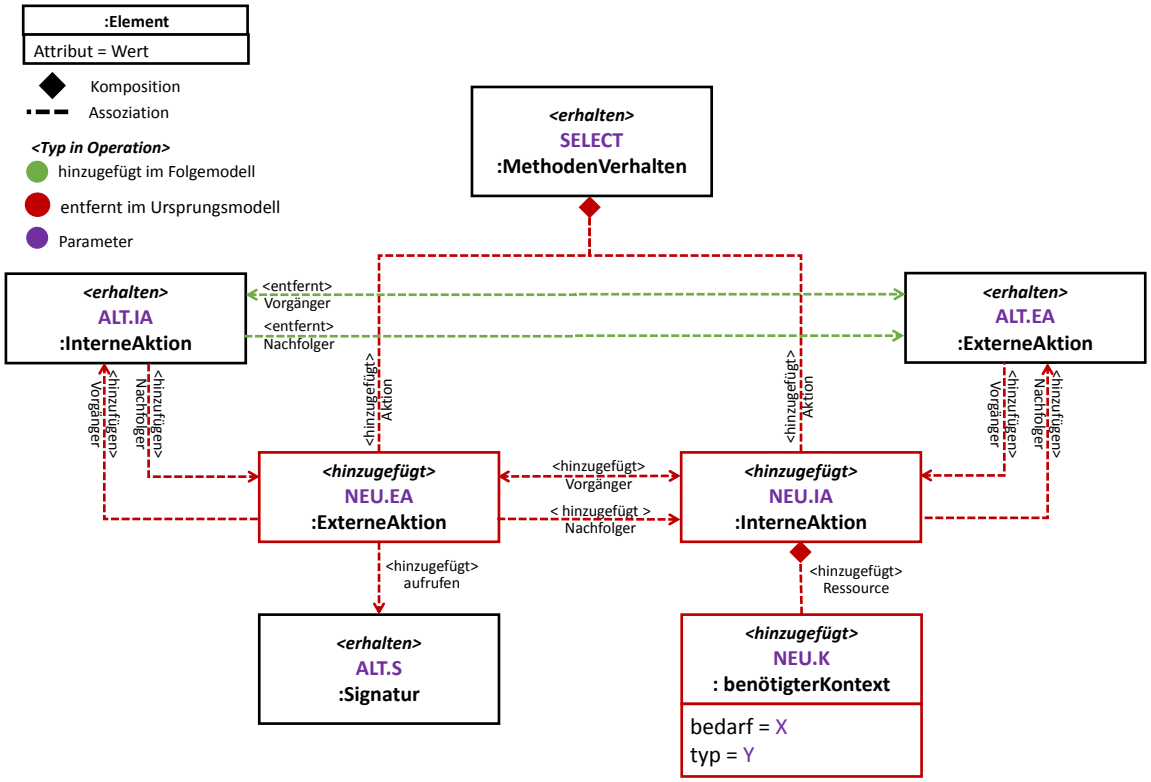


Abbildung 6.23: Operation A: Einfügeaktion

Die primitiven Änderungen gleichen dem Entfernen der Operation A (Einfügeaktion), sowie der Operation G (Zeit) zur Modifizierung der Zeitbedarfs. Die wesentliche Operation A ist in Abbildung 6.23 gezeigt. Diese entfernt zwei vorhandenen Aktionen und ihre jeweiligen Assoziationen.

Entsprechend ergibt sich das Editierskript, welches nur aus diesen Operationen besteht. Eingangsparameter werden an die gefundenen Elemente gebunden ($SELECT$, $ALT.IA$, $ALT.EA$ und $ALT.S$) und entsprechende Ausgangsparameter sind die entfernten Elemente ($NEU.EA$ und $NEU.IA$). Dabei werden die Wertparameter (X und Y) mit den konkreten Werten der Attribute aus $\tilde{m}$ parametrisiert. Dabei sind die Element- und Wertparameter nicht beschränkt, d.h. das sie mit beliebigen konkreten Werten belegt werden können. Das Editierskript $\delta(m, \tilde{m})$ hat keine Abhängigkeiten und kann auf jedes Modell mit zugeordneten Parametern angewendet werden.

Der Modellkontext besteht aus dem Kontext der Eingangsparameter $ALT.IA$ und $ALT.EA$. Diese umfassen neben dem Zeitkontext des Elements *InterneAktion* auch die zu vergleichende Signatur $ALT.S$. Über den Typ- und Ortskontext dieser Signatur und ihre Lage in der betrachteten Aufruffolge kann identifiziert werden, welche Methoden entfernt werden sollen.

[49] In Abbildung 6.22 vernachlässigt.

Für das **Ergebnis** müssen die nichtfunktionalen Eigenschaften bestimmt werden. Dies wird am Beispiel der Methode *KA.barcodeScanned* gezeigt. Dazu sind nach Verfahrensvorschrift 5 die Mittelwerte von allen assoziierten *BenötigterKontext*-Elementen und iterativ die aufgerufen Methoden zu summieren. Daraus ergibt sich die Performanzmetrik der mittlere Antwortzeit:

$$f_{analyse}^{Performanz} = \frac{1}{1,2+4,9+2,2+2,4+2,0+2,5+0,8} = 0,0625 \tag{6.12}$$

Die Methode tätigt drei Aufrufe, wodurch sich die Kopplungen bezüglich der Aufrufe ergibt. Aufgerufen wird die betrachtete Methode nur durch die Lebenszyklusmethode der Komponente *BarcodeScanController*. Damit ist die Wartbarkeitsmetrik wie folgt:

$$f_{analyse}^{Wartbarkeit} = \frac{1}{\begin{array}{l}|\{B.getProdForCode, BD.runningChanged,\\ KG.runningChanged\} \cup \{BC.lebenszyklus\}|\end{array}} = 0,25 \tag{6.13}$$

Mit den Eigenschaftswerten lässt sich das EffektAn-Wertetypobjekt im Vergleich zu den identisch bestimmten Eigenschaften der *KA.addLastProduct* Methode bilden. Diese sind in Tabelle 6.23 angegeben:

	Performanz	Wartbarkeit	Performanz	Wartbarkeit	Performanz	Wartbarkeit
Model	**Werte**		**AbsDiff**		**RelDiff**	
barcodeScanned	$0,0625$	$0,25$	$0,0275$	$0,08$	44%	33%
addLastProduct	$0,09$	$0,33$				

Tabelle 6.23: Eigenschaften des Effekts für *KA.barcodeScanned* und *KA.addLastProduct*

In diesem Experiment wird die **Bewertung** nicht genutzt, um den Evolutionsschrittes zu bewerten, sondern um die Ähnlichkeit zu quantifizieren:

- Entsprechend wird für den **Grund** die veränderte Metrik in Formel 6.14 genutzt. Diese setzt die Anzahl der abweichenden Aktionselement aus der Menge aller Abweichungen $ABW = \{abw_1, \cdots, abw_i, \cdots abw_n\}$[50] zu der Gesamtanzahl n von vorhandenen Aktionselementen und der Startaktion des Modells in Beziehung.[51]

  $$f_{Grund} = \frac{\sum_{i=1 \in ABW} abw_i \cdot bewertung_e^{abw}}{n+1} \tag{6.14}$$

  Für das Experiment wird die Klasse *Wegfallen* stärker bewertet.[52]. Darunter ergibt sich die Anwendungsbewertung des Grunds:

  $$f_{Grund} = 1 - \frac{2 \cdot 1 + 1 \cdot 0,5}{8} = 0,6875 \tag{6.15}$$

  Diese Metrik quantifiziert wie viele Abweichungen im Verhältnis zur Gesamtgröße des Modells beobachtet wurden.

[50] Es wird nur die erste Abweichung eines jeden Aktion-Elements berücksichtigt.

[51] $bewertung^{abw}$ gibt weiterhin die Bewertung der Klasse der Abweichung an.

[52] Es werden die Bewertungen $bewertung_{Wegfallen}^{abw} = bewertung_{Hinzufügen}^{abw} = 1$ und $bewertung_{Zeitaberration}^{abw} = bewertung_{Veränderung}^{abw} = 0,5$ verwendet.

- Für ein Ähnlichkeitsmaß der **Reaktion** ist entscheidend, wie die Parameter der Operationen des Editierskripts an das Ursprungsmodell gebunden werden. Für das Editierskript gibt es dabei zwei Fälle: Im ersten Fall handelt es sich um eine Aktion, die durch eine vorherige Operation erzeugt wurde. Es gibt somit keine Zuordnung zwischen den Varianten und es besteht keine Ähnlichkeit der Aktionen ($f_{\ddot{A}hnlichkeit}(a, \tilde{a}) = 0$). Im zweiten Fall einer Zuordnung ist dessen Ähnlichkeit entscheidend. Dies ist für die betrachten Varianten der Fall. Die zwei Parameter wurden mit einer Ähnlichkeit von 1 zugeordnet, da die Aktionen identisch sind. Damit ergibt sich die Anwendungsbewertung der Reaktion:

$$f_{Reaktion} = \frac{1+1}{2} = 1 \tag{6.16}$$

  Diese Metrik gibt darüber Auskunft, wie gut der gebildete Evolutionsschritt angewendet werden kann und damit wie ähnlich sich die zwei Modelle sind.
- Das **Ergebnis** gibt die relative Differenz von nichtfunktionalen Eigenschaften an. Diese sollten in ähnlichen Modellen möglichst gleich sein. Entsprechend wird die Bewertung invertiert, wobei eine Verschlechterung oder Verbesserung keine Rolle spielt. Mit den relativen Differenzen der Performanz von 44% und der Wartbarkeit von 33,33% ergibt sich eine Bewertung des Ergebnisses von $f_{Ergebnis} = 1 - \frac{|-0,44| + |0,3333|}{2} = 0,6133$. Diese Metrik wiegt somit die Eigenschaften gegeneinander auf und überprüft, ob ein Ungleichgewicht bei den berechneten Varianten besteht.

Unter einer Gleichgewichtung lässt sich Bewertung des Evolutionsschrittes und damit die Ähnlichkeit berechnen:

$$f_{bewertung}(\Delta(m, \tilde{m}), \hat{m}) = \frac{1 \cdot 0,6875 + 1 \cdot 1 + 0,615 \cdot 1}{3} = 0,7675$$

Die große Ähnlichkeit wird nur aufgrund des Wegfalls von einem Aufrufen und einer entsprechenden Veränderung der nichtfunktionalen Eigenschaften reduziert. Der Verhaltensunterschied kann dabei auch durch die Anwendung des Evolutionsschrittes modellbasiert beschrieben werden, indem unter den Zuordnungen $s \leftrightarrow s$, $ea_1 \leftrightarrow ea_1$ und $ia_1 \leftrightarrow ia_1$ die Handlungsalternative $m^{\Delta(m,\tilde{m})}$ erzeugt wird.

Diskussion der Ergebnisse

Für das Experiment wurden alle Methodenverhaltensmodelle in einer wissenstragenden Komponente miteinander verglichen. Tabelle 6.24 zeigt das über Evolutionsschritte berechnete Ähnlichkeitsmaß dieser Methoden. Dabei sind sich eine Reihe von Methoden sowohl in ihrem Ergebnis von Eigenschaften, als auch beim Grund und der Reaktion des Evolutionsschrittes jeweils ähnlich. Die Methoden, welche in den Evolutionsschritten zugeordnet werden, wie z.B. die *runningChanged*-Methoden verschiedener Komponenten, haben die höchste Ähnlichkeit. In diesen Fällen tritt nur eine Abweichung der Klasse Zeitaberration auf. Vergleichbares gilt für die verschiedenen Varianten der Methoden, die nur Informationen austauschen (*runningChanged*, *changeAmount* und *cashEntered*). Diese haben insbesondere eine vergleichbare Antwortzeit (Performanz) und Kopplung (Wartung). Gleiches gilt für die Methoden *saleFinished* und *saleSuccess*, welche jedoch eine abweichende Performanz aufweisen. Weitere vergleichbare Methoden sind $cashBox$ und *paymentMethod*, da die erste Methode durch die zweite Methode aufgerufen wird und die Performanz durch die Interaktion mit dem Kassierer maßgeblich bestimmt ist. Die geringste Ähnlichkeit besteht bei der Lebenszyklusmethode des gesamten Verkaufsprozesses *saleStart* und der Methode *cashEntered*, die eine Vielzahl unterschiedlicher Methoden aufruft.

Die zuvor detailliert betrachteten Methoden *KA.barcodeScanned* und *Bestand.getProductForCode* haben in beide Richtung eine hohe Ähnlichkeit. Die Differenz der Betrachtungsrichtung basiert auf der Asymmetrie des Evolutionsschrittes beim Ergebnis. Denn die relative Differenz wird auf Grundlage der ursprünglichen Eigenschaftsausprägung berechnet. Dies ist für das Ähnlichkeitsmaß als nicht optimal, allerdings tolerierbar einzustufen.

	KG.runningChanged	KG.changeAmount	KA.barcodeScanned	KA.saleStart	KA.saleFinished	KA.cashEntered	KA.paymentMode	KA.addLastProduct	BD.runningChanged	BD.changeAmount	BD.saleFinished	BD.saleSuccess	GK.cashBox	GK.cashEntered	GK.changeAmount	B.getProductForCode	B.updateInventory
KG.runningChanged	1,00	0,61	0,24	0,15	0,40	0,20	0,38	0,34	0,77	0,67	0,41	0,39	0,39	0,62	0,65	0,43	0,42
KG.changeAmount	0,68	1,00	0,20	0,12	0,34	0,21	0,28	0,25	0,69	0,77	0,32	0,40	0,27	0,60	0,71	0,37	0,40
KA.barcodeScanned	0,21	0,11	1,00	0,15	0,32	0,38	0,47	0,77	0,42	0,20	0,20	0,27	0,45	0,22	0,20	0,28	0,37
KA.saleStart	0,17	0,15	0,24	1,00	0,20	0,27	0,16	0,24	0,17	0,15	0,13	0,15	0,18	0,18	0,17	0,14	0,16
KA.saleFinished	0,31	0,26	0,39	0,11	1,00	0,33	0,27	0,39	0,31	0,29	0,64	0,40	0,27	0,30	0,29	0,29	0,30
KA.cashEntered	0,14	0,08	0,31	0,17	0,23	1,00	0,42	0,14	0,14	0,13	0,10	0,11	0,44	0,13	0,08	0,12	0,26
KA.paymentMode	0,33	0,27	0,33	0,25	0,27	0,40	1,00	0,33	0,32	0,28	0,30	0,10	0,76	0,27	0,26	0,30	0,11
KA.addLastProduct	0,49	0,28	0,80	0,17	0,43	0,28	0,36	1,00	0,50	0,32	0,37	0,32	0,35	0,30	0,28	0,32	0,36
BD.runningChanged	0,77	0,60	0,28	0,18	0,38	0,20	0,34	0,29	1,00	0,66	0,41	0,38	0,37	0,59	0,64	0,43	0,46
BD.changeAmount	0,67	0,74	0,22	0,12	0,33	0,23	0,31	0,30	0,65	1,00	0,31	0,41	0,30	0,61	0,73	0,38	0,44
BD.saleFinished	0,34	0,25	0,29	0,11	0,68	0,22	0,26	0,37	0,34	0,27	1,00	0,74	0,26	0,30	0,28	0,38	0,40
BD.saleSuccess	0,40	0,37	0,28	0,16	0,46	0,22	0,16	0,39	0,42	0,36	0,84	1,00	0,19	0,35	0,35	0,41	0,36
GK.cashBox	0,36	0,28	0,35	0,24	0,28	0,41	0,77	0,37	0,36	0,25	0,29	0,10	1,00	0,26	0,28	0,30	0,11
GK.cashEntered	0,56	0,56	0,25	0,10	0,32	0,20	0,29	0,28	0,57	0,60	0,33	0,40	0,28	1,00	0,55	0,36	0,40
GK.changeAmount	0,65	0,73	0,20	0,12	0,32	0,18	0,26	0,27	0,60	0,72	0,36	0,43	0,29	0,61	1,00	0,32	0,45
B.getProductForCode	0,23	0,20	0,20	0,02	0,25	0,15	0,18	0,27	0,26	0,18	0,28	0,25	0,19	0,21	0,16	1,00	0,36
B.updateInventory	0,39	0,36	0,43	0,12	0,44	0,33	0,18	0,42	0,36	0,35	0,34	0,31	0,20	0,37	0,38	0,35	10

Tabelle 6.24: Ähnlichkeitsmatrix der Methoden der CBCPS-Fallstudie auf Basis der Bewertungen von Evolutionsschritten

Aus dem Experiment zum Ähnlichkeitsmaß von Varianten lassen sich folgende Schlussfolgerungen ableiten:

- Das Konzept eines schrittbasierten Evolutionsprozesses kann auch auf etablierte Modelle, wie z.B. PCM-Performanzmodelle, angewendet werden.
- Wesentlicher Nachteil bei der Methodik für CBCPSe ist die fehlende Abbildung von internem Verhalten der Methoden. Dies könnte für Evolutionsschritte aufgelöst werden indem beispielsweise Entwicklungsmodelle eingesetzt werden, die dieses Verhalten abbilden. Dies würde allerdings der Grundannahme der Arbeit widersprechen, Modelle nur mittels externen Verhaltens zu koevolvieren. Dennoch könnten solche Methoden in die Evolutionsunterstützung integriert werden.
- Evolutionsschritte sind auch in anderen Anwendungsbereichen, wie z.B. bei der Bewertung von Varianten, einsetzbar. Dabei ermöglichen sie jeweils Vergleiche, die alle wesentlichen Dimensionen von Änderungen umfassen. Zusätzlich wird dabei die Verhaltensähnlichkeit in Form von Modellen beschrieben, sodass die Transformation von einer Variante in die andere Variante aufgezeigt wird.

Zusammenfassende Bewertung des fünften Evaluationsziels

Das Experiment hat gezeigt, dass alle Aspekte des Grunds, der Reaktion, als auch des Ereignisses Möglichkeiten bieten die Ähnlichkeit von Varianten zu beschreiben. Entsprechend konnte für Evolutionsschritte ein Ähnlichkeitsmaß definiert werden, dass unterschiedliche Aspekte der Ähnlichkeit betrachtet, diese quantifiziert und durch auftretende Abweichungen, asymmetrische Differenzen und Eigenschaftsveränderungen darstellbar macht. Über ein solches Ähnlichkeitsmaß ist es möglich den Unterschied über die verschiedenen Aspekte hinweg zu bewerten, diese den Nutzerpräferenzen entsprechend gegeneinander abzuwiegen und in Form von Modelldifferenzen auch visuell darzustellen.

6.4 Zusammenfassung

Für die Evaluation wurden in Abschnitt 6.1 fünf Ziele definiert. Zur praktischen Untersuchung der Zielerreichung wurden anschließend zwei unterschiedliche Fallstudien vorgestellt, die mit CPPSen und CBCPSen jeweils zwei unterschiedliche Anwendungsdomänen haben. Die Fallstudien unterschieden sich weiterhin in sequenziellen Ausbaustufen und parallel existierenden Varianten sowie Anwendungszielen bzw. -schwerpunkten. Dadurch kann ein repräsentativer Ausschnitt von CPSen betrachtet werden (Abschnitt 6.2).

Für die Ziele wurde dann – wie in Abschnitt 6.3 beschrieben - in einer Reihe von Experimenten die Umsetzbarkeit der vorgeschlagenen Konzepte in einer Machbarkeitsstudie gezeigt und es wurden die dabei gewonnene Ergebnisse diskutiert. Mit der Anwendung der Beschreibung von Evolutionsschritten wurde dabei u.a. gezeigt, dass auf diese Weise sowohl erhaltende als auch verändernde Evolution erfasst werden kann. Dies ermöglichte es auch, die Evolution verschiedener Typen von CPSen zu kategorisieren.

In vier weiteren Experimenten wurde dann anhand der im Rahmen dieser Arbeit vorgenommenen Implementierung gezeigt, dass Zustände konsistent und performant extrahiert werden können. Dies gilt sowie für sequenzielle Zustände eines CPPS-Bussystems als auch für instrumentalisierte Dienstaufrufe von CBCPSen. Dies zeigt auch, dass die Systemarchitektur in heterogenen Anwendungskontexten verwendet werden kann. Zusätzlich wurden über eine statische Analyse des Quellcodes Kontexteinformationen für Methoden eines CBCPSs extrahiert. Dies zeigt das vorhandene Methoden, wie z.B. der Auswirkungsanalyse oder Modellextraktion, Teil der in dieser Arbeit entworfenen Methodik sein können. Und weiterhin konnte mit der Simulation von Testfällen gezeigt werden, dass auch Entwicklungsmodellen als Grundlage genutzt werden können. Damit konnte die Integration von unterschiedlichen Quell- und Kontextartefakten erfolgreich demonstriert werden.

Mit den verschieden Modellartefakten wurde gezeigt, dass eine Koevolution von Laufzeitmodellen durch die wissenstragende Komponente erreicht wird. Dabei konnten für die Fallstudie von CPPSen Evolutionsschritte gebildet werden, die es ermöglichen, den Evolutionsprozess von CPPSen abzubilden. Anhand mehrerer wissenstragender Komponenten wurde so gezeigt, dass eine sinnvolle Unterstützung durch Vorschläge von Handlungsalternativen angeboten werden kann - auch wenn dabei bezüglich der erfassbaren Semantik und der prototypischen Implementierung der wissenstragenden Komponente noch Grenzen aufgezeigt wurden.

Der erweiterbare Anwendungsfokus der Arbeit konnte durch die Übertragung von Evolutionsschritten auf ein Ähnlichkeitsmaß für Varianten gezeigt werden. In einem Experiment der Fallstudie für CBCPSe wurde gezeigt, dass dadurch Ähnlichkeiten der unterschiedlichen Aspekte von Evolution operationalisiert und quantifiziert werden können.

7 Schlussbetrachtung

Im folgenden, abschließenden Kapitel wird die im Rahmen dieser Arbeit zugrunde gelegte Methodik und vorgeschlagene Systemarchitektur noch einmal zusammenfassend betrachtet. Abschnitt 7.1 fasst dazu zunächst die praktischen Ergebnisse der Arbeit zusammen. Die wesentlichen Forschungsbeiträge dieser Ergebnisse werden dann im darauffolgenden Abschnitt 7.2 herausgestellt. Im abschließenden Abschnitt 7.3 werden mögliche weitere Fragestellungen und methodische sowie operative Erweiterungen, die auf Ergebnissen der Arbeit aufbauen können, dargelegt.

7.1 Ergebnisse der Arbeit

Diese Arbeit identifiziert den Mangel und die Notwendigkeit einer kohärenten Methode und Systemarchitektur, die modellbasierte Methoden für die Systembeschreibung so miteinander verknüpft, dass Wissensartefakte während ihrer Evolution bewahrt, weiterentwickelt und verwendet werden können [GRG+15]. Angesichts der Interdisziplinarität von langlebigen cyber-physischen Systemen sind für diese Koevolution u.a. modellbasierte Artefakte mit Laufzeitinformationen zu verknüpfen [VHFF+15], die durch das cyber-physische System selbst verwaltet und verarbeitet werden können [ZT16]. Über eine verteilte, nichtinvasive Systemarchitektur können diese Evolutionsartefakte dann sowohl deskriptiv als auch über Vergleiche präskriptiv zur Evolutionsunterstützung genutzt werden [LBK15].

Dem folgend versteht die Arbeit Wissen über die Evolution eines Systems als dem System inhärente, wiederverwendbare Erfahrungen. Diese werden über extrahierte Artefakte im Netzwerk eines cyber-physischen Systems u.a. für die Dokumentation und Fortführung des Evolutionsprozesses bereitgestellt. Aus dieser Sichtweise heraus stellt die Arbeit ein Konzept vor, das Evolution als Prozess von erfassbaren, modellbasierten Evolutionsschritten modelliert, die als Evolutionsartefakte in einer unabhängig wissenstragenden Komponente zur Laufzeit verwaltet werden. Die dafür notwendigen Aktivitäten zeigt Abbildung 7.1 zusammenfassend unter der Zuordnung von wesentlichen Architekturkomponenten.

Auf unterster Abstraktionsebene der cyber-physischen Verbindung integrieren Adapterkomponenten einer wissenstragenden Komponente unterschiedliche Ereignisquellen des technischen Prozesses in die Evolutionsunterstützung, indem diese von außen bei ihrer Ausführung beobachtet werden (*Einbinden von Quellen*). Zur Interpretation der Ereignisse werden dabei Kontextinformationen über zusätzliche Adapter gewonnen oder durch den menschlichen Nutzer hinzugefügt. Die Adapterkomponenten stellen dabei aus methodischer Sicht Quell- bzw. Kontextartefakte für Bussysteme, Dienstaufrufe, Testmodelle oder den Softwarecode bereit.

Die Zustände des cyber-physischen Systems werden anschließend in einer Hierarchie von Repräsentationskomponenten aus den Ereignissen erzeugt und konsistent gehalten. Diese können als digitaler Zwilling von Komponenten des cyber-physischen Systems verstanden werden, die jeweils *Zustände in Informationsartefakten* für eine weitere Verarbeitung in der wissenstragenden Komponente bereitstellen. An dieser Stelle wird u.a. für eine bessere Akzeptanz von der Abbildung code-internen Verhalten abgesehen als auch von harten Echtzeitanforderungen abstrahiert.

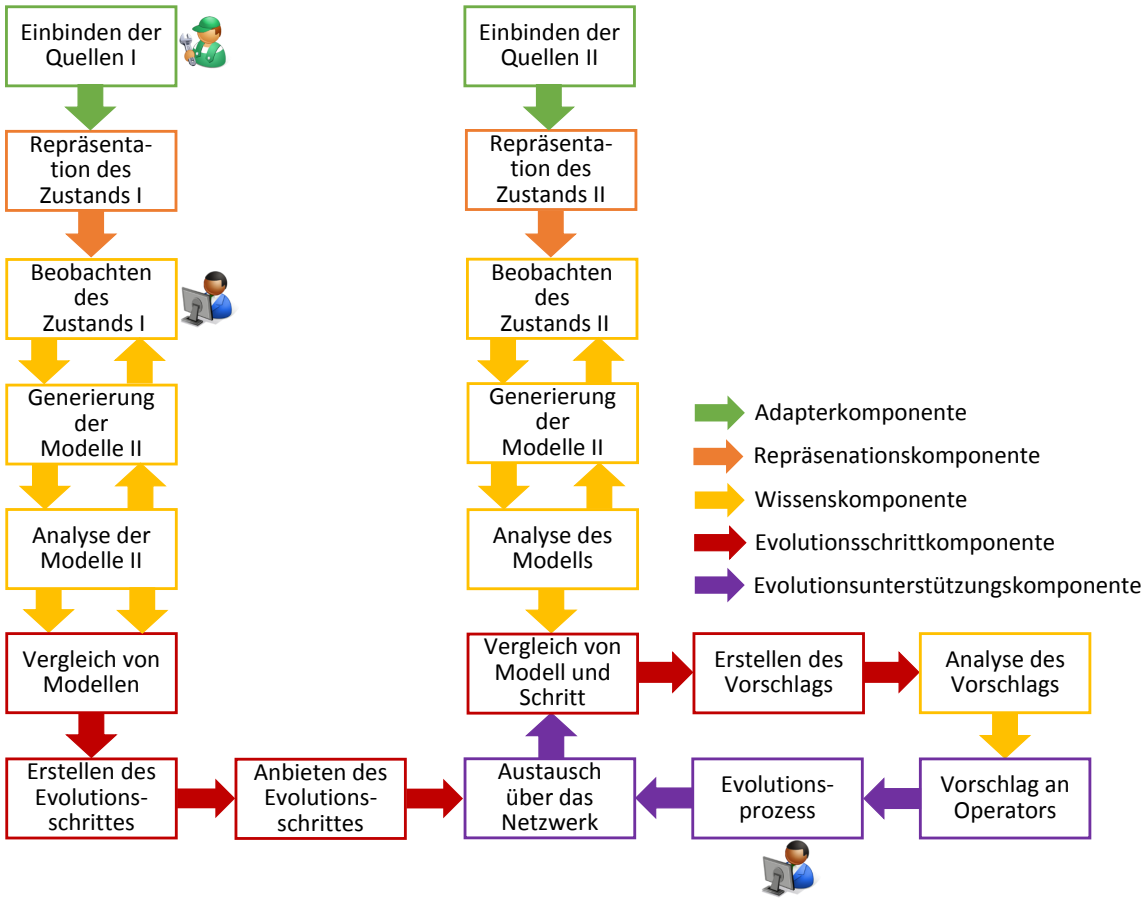


Abbildung 7.1: Aktivitäten der Evolutionsunterstützung unter Zuordnung wesentlicher Architekturkomponenten

Durch die Wissenskomponenten werden Modellartefakte implementiert und softwaretechnisch über den Zustandsdienst der Repräsentationskomponenten gekapselt. Die Wissenskomponenten führen dabei selbstständig einen dienstbasierten Prozess aus, um die Koevolution ihrer Modelle zu gewährleisten. Dazu wird erstens der *Zustand stetig beobachtet* und es werden Abweichungen zwischen diesen Zuständen und den aktuellen Modellen erkannt. Für eine Entscheidungsfindung bei diesen Abweichungen wird der menschliche Nutzer eingebunden. Zweitens werden *Modelle* aus diesen Zuständen *generiert*. Und diese Modelle werden drittens bezüglich nichtfunktionaler Eigenschaften *analysiert*. Durch diesen autonom gesteuerten Prozess der Koevolution erzeugen die Wissenskomponenten iterativ Versionen der Modelle. Diese Versionen bilden in Abhängigkeit zum Modelltyp jeweils Verhalten für eine bestimmte Ausprägung von cyber-physischen Systemen, wie z.B. cyber-physische Produktionssysteme oder cloud-basierte cyber-physische Systeme, ab.

Um einen schrittbasierten Evolutionsprozess zu erhalten, werden in der Evolutionsschrittkomponente diese *Modelle miteinander verglichen*. Dazu werden über ein ähnlichkeitsbasiertes Verfahren zunächst die Elemente der Modelle einander zugeordnet. Um einen *Evolutionsschritt zu erstellen*, wird das System als Besitzer spezifiziert, die auslösende Abweichung als Grund klassifiziert, die Verhaltensänderung als modellbasierte Reaktion abbildet und die Ergebnisse der Evolution durch Eigenschaften der Analyse von Modellen quantifiziert. Für die Bestimmung dieser Aspekte werden dafür Verfahren für die Erkennung von Abweichungen, die Differenzenberechnung und die Modellanalyse in die Evolutionsunterstützung integriert.

Um darüber hinaus auch eine Unterstützung von anderen Systemkomponenten in einem verteilten cyber-physischen System zu ermöglichen, bietet die Evolutionsschrittkomponente der wissenstragenden Komponente alle abgeleiteten Evolutionsschritte auf Ebene des Cyber-Netzwerks in einem verteilten, blockchain-basierten Marktplatz an. Wenn hier eine weitere wissenstragende Komponente einen Evolutionsschritt nutzen will, wird das entsprechende vorliegende, aktuelle Modell dieser Komponente mit dem Modellkontext des Evolutionsschrittes *verglichen*. Denn die realisierte Evolutionsunterstützung erlaubt durch ihre unterliegende Modelltransformation die Übertragung der Wirkung des Evolutionsschrittes auf das vorliegende Modell. Dies ermöglicht es auf Modellebene, einen entsprechenden *Vorschlag für den Nutzer zu erstellen*. Dieser Vorschlag enthält neben einer Handlungsalternative auch die vorausgesagten Eigenschaftsveränderungen, die aus den *Analyseergebnissen der abgeleiteten Handlungsalternative* und den im Marktplatz bereits vorliegenden Erfahrungen des genutzten Evolutionsschrittes bestehen.

Dadurch können in der wissenstragenden Komponente eine passive und aktive Suche als auch der Austausch und die Bewertung von Evolutionsschrittartefakten realisiert werden. Der bewertete Vorschlag wird dem Nutzer angeboten und kann entsprechend der Handlungsalternative durch diesen implementiert werden. Die damit erreichte Fortführung des Evolutionsprozesses wird bezüglich der zeitlich-räumlichen Verknüpfung und der Vererbung von Erfahrungen in der wissenstragenden Komponente dokumentiert. Dies ermöglicht eine stetige Evolutionsunterstützung, indem Änderungen in Evolutionsschritten abgebildet werden und potentiell sinnvolle Evolutionsschritte dem Nutzer auf dieser Grundlage automatisiert vorgeschlagen werden.

7.2 Forschungsbeiträge

In dieser Arbeit werden unterschiedliche Teilbereiche der Softwaretechnik und verwandter Anwendungsbereiche miteinander verknüpft, um einen Beitrag zu Problemstellungen der Evolution von cyber-physischen Systemen zu leisten. Nachfolgend werden diese Forschungsbeiträge entlang der Veröffentlichungen des Autors[1] zusammenfassend dargestellt. Jede Veröffentlichung wird dabei einmalig an der Stelle ihres jeweiligen thematischen Schwerpunktes zitiert.

Der zentrale Forschungsbereich der Arbeit ist die ganzheitliche Betrachtung der Gewinnung, Bündelung, Verarbeitung und Nutzung von Artefakten im Rahmen der Softwareevolution. Dies umfasst zum einen die Schaffung eines einheitlichen Konzepts zur Beschreibung der Evolution, sowie dessen Umsetzung in einer kohärenten Methodik von modellbasierten Entwicklungs-, Beobachtungs- und Verarbeitungsverfahren. Und zum anderen beinhaltet es die Integration und Ausführung dieser Verfahren in einer selbstwahrnehmenden, verteilten und leicht integrierbaren Systemarchitektur, die den spezifischen Bedingungen eines cyber-physischen Systems gerecht wird.

Bei der **Problemstellung und Beschreibung von evolutionären Änderungen** konnten u.a. mit dem Mangel an Evolutionsartefakten, einer kohärenten Methodik und der fehlenden Fortführung für selbstwahrnehmende cyber-physische Systeme Problemstellungen des zentralen Forschungsbereiches identifiziert und herausgearbeitet werden ([HLLF14a]). Dadurch konnte ein Erkenntnisgewinn für spezifische Probleme der Evolution sowie für die Nutzung von Synergien zwischen vorhandenen Lösungsverfahren von Einzelproblemen aufgezeigt und zueinander in Verbindung gebracht werden ([VHFF+15]).

[1] Die Veröffentlichungen sind am Ende des Kapitels noch einmal gesondert aufgeführt.

Im Rahmen der vorliegenden Arbeit wurde ein neuartiges Konzept von Evolutionsschritten entwickelt, das Änderungen in allen aus der Literatur identifizierten Aspekten beschreibbar macht. Für die wesentlichen Aspekte wurde dabei mit dem CRI-Modell eine allgemeingültige Beschreibung in sechs Dimensionen vorgeschlagen, die für spezifische Systeme mit abstrakten Domänenmodellen kategorisiert werden kann ([RPH+17]). Dieses Modell bringt die Bereiche der Anomalieerkennung und der Softwareevolution zusammen und ermöglicht es damit, sowohl kurzfristige Ad-hoc-Änderungen als auch langfristig geplante Änderungen in einer einheitlichen Beschreibung zu spezifizieren und operativ zu verwenden ([HPR+18]).

Diese Beschreibung unterschiedet sich von vergleichbaren Ansätzen u.a. durch ihren Fokus auf durch das System selbst beobachtbares Verhalten und die Anwendbarkeit der identifizierten Änderungen auf der Modellebene. Dadurch ermöglicht das vorgestellte Konzept die Verknüpfung in einem Evolutionsprozess, der beobachtbare Änderungen als anwendbare Modelldifferenzen versteht und dadurch unentdeckte Potentiale und Konfigurationen aus vergleichbaren Systemen aufdecken kann ([HCL+17]). Diese Art des Vorgehens ist an die Forschungen zu (Software-)Produktlinien angelehnt und wurde im Rahmen dieser Arbeit erstmals auf koevolvierende Laufzeitartefakte angewendet.

Die damit zu realisierende **Koevolution von System und Modell** baut methodisch auf vorhandenen Ansätzen der Beobachtung von allgemeinen Softwaresystemen und echtzeitgesteuerten cyber-physischen Systemen auf und integriert bzw. überträgt als Forschungsbeitrag diese in die konzeptionelle Sichtweise der Evolution ([HWB+13]). Dabei konnten in unterschiedlichen softwaretechnischen Forschungsbereichen, wie u.a. dienstorientierten Architekturen, Testverfahren von Softwareproduktlinien oder modellbasierte Entwicklungsverfahren, Methoden und Synergien bezüglich der entsprechenden Laufzeitartefakte beigetragen werden ([HLL+14, LFH+15]).

Die Methodik dieser Arbeit verwendet dabei eine domänenunabhängige Beschreibungsform und beschränkt sich softwaretechnisch auf einen nichtinvasiven Ansatz. Dies ermöglicht es, bei der strukturellen Integration von Modellartefakten methodisch von den verwendeten Modelltypen und dem unterliegenden System zu abstrahieren. Entsprechend können die vorgeschlagenen Verfahren auch auf andere Modelltypen und Systeme angewendet werden und so den Anwendungsbereich der Unterstützungslösung erhöht wird. Entsprechende Beiträge konnten dabei im Rahmen der Forschungstätigkeit dieser Arbeit mit spezifischen neuen Modellen und bei etablierten Modellen geleistet werden ([LFA+15, LHFL15]).

Die Koevolution wurde mit einem halbautomatisierten Prozess realisiert, durch den Systeme autonom und systematisch Modelle generieren, Abweichungen feststellen und nichtfunktionale Eigenschaften analysieren können ([LHFL14a, LHFL14b]). Dieser Prozess ist dabei allgemein anwendbar, sodass er auch in vergleichbaren Forschungsbereichen eingesetzt werden kann, in denen eine Koevolution von System und Modell notwendig ist. Diesbezüglich trägt diese Arbeit damit auch zu Ergebnissen in den Anwendungsgebieten von cyber-physischen Systemen bei, indem beispielsweise Eigenschaften von Produktionsanlagen im Rahmen von Forschungskooperationen operationalisiert wurden ([LWA+13, LFHL13]).

Mit der **Bildung von Evolutionsschritten** aus Abweichungen, Modelldifferenzen und Eigenschaften wurde weiterhin eine softwaretechnische Repräsentation von Änderungen entlang von digitalen Zwillingen eines cyber-physischen Systems eingeführt. Diese ermöglicht es, alle Aspekte, insbesondere den Grund, die Reaktion und das Ergebnis einer Änderung, abzubilden und diese operativ als Artefakte erfass- und so auch verarbeitbar zu machen. Dabei wurden aktuelle Modelltransformationskonzepte auf Evolutionsschritten angewendet und in einem veränderten Kontext erprobt und verbessert ([PKH+18, CHFL18]).

Mit der Anwendung auf vergleichbare Modelle von cyber-physischen Systemen wurde eine neue Form eines modellbasierten Vorschlagsystem erforscht, die einzig auf dem Potenzial von in cyber-physisch Systemen vorliegenden, horizontal integrierten Komponenten beruht ([AKT+18]). Die dadurch erreichte Evolutionsunterstützung eröffnet neuartige Chancen des Lernens aus verteilt vorliegenden Erfahrungen anderer ähnlicher Komponenten und ermöglicht es dem Nutzer, in einer für ihn verständlicher Form mögliche Handlungsalternativen zu erhalten und zu bewerten ([HBL+18]).

Die Arbeit entwickelte schließlich eine **Systemarchitektur**, die über eine externe Beobachtung den Anforderungen von cyber-physische Systemen gerecht wird. Dabei konnte mit der Integration von heterogenen Informations- und Kontextquellen, der Verwaltung von domänenspezifischen Laufzeitmodellen und ihrer Eigenschaften sowie einem kooperativen Austausch zwischen verteilten Komponenten ein Beitrag zur Evolutionsforschung für cyber-physischen Systemen erbracht werden ([HLLF14b]). Darauf basierend wurde im Rahmen dieser Arbeit eine prototypische wissenstragende Komponente realisiert, die eine einheitliche Repräsentation von Verhalten zur Laufzeit, die Anreicherung dieses Verhaltens mit Kontext und den Austausch der entstandenen Artefakte auf Basis moderner Technologien implementiert. Dies umfasst u.a. ein neuartiges Konzept für eine strategieorientierte Auswertung von modellspezifischen Aussagen über einen Zustandsdienst sowie die Kapselung von Artefakten in Laufzeitkomponenten ([HLF18]). Ein komponentenübergreifender Austausch wurde dabei als neuartiger Anwendungsbereich einer verteilten Blockchain-Technologie realisiert.

Durch die Separierung der verschiedenen Ebenen eines cyber-physischen Systems mit in sich gekapselten, autonomen Komponenten kann die Systemarchitektur unabhängig vom Ziel der Evolution auch im Bereich der selbstwahrnehmenden und selbstadaptiven Systeme eingesetzt werden. Dadurch wurde mit dieser Systemarchitektur auch in verwandten Bereichen ein Forschungsbeitrag geleistet. Dies umfasst u.a. eine flexible Prozessausführung beispielsweise für selbstorganisierte Systeme ([HVL13]) als auch die Erkennung von sicherheitsrelevanten Anomalien zur Laufzeit ([JSB+18]) sowie die Ausführung von zielorientierten Prozessen über Belief-Desire-Intention-Agenten ([PBHL14]).

Zusammenfassend stellt die Arbeit damit eine übergreifende Vision, eine darauf aufbauende Konzeption und auch eine konkrete (prototypische) Implementierung einer modellbasierten Evolutionsunterstützung für cyber-physische Systeme vor. Sie erweitert damit auch allgemeine Verfahren, Modelle und Architekturkonzepte für angrenzende Forschungsbereiche.

7.3 Ausblick auf weitere Forschung

Die in dieser Arbeit vorgestellte Konzeption bietet darüber hinaus Anknüpfungspunkte und Erweiterungen für zukünftige Forschungsarbeiten, wovon einige folgend vorgestellt werden.

Die **Beschreibung von Evolutionsschritten** wurde auf Grundlage der beiden Forschungsbereiche von cyber-physische Produktionssystem und cloud-basierten cyber-physischen Systemen vorgenommen. Entsprechend wäre es interessant, eine Verifizierung und Erweiterung der Beschreibung in anderen Bereichen von (verteilten) Systemen vorzunehmen. Dabei bietet insbesondere die Abbildung des Ergebnisses eines Evolutionsschrittes noch Potenzial für Erweiterungen, da diese hauptsächlich auf nichtfunktionale Eigenschaften und insbesondere Performanzmetriken fokussiert. Neben diesen könnten auch weiterer Metriken, beispielsweise bezüglich der Komplexität oder Zukunftsfähigkeit von Systemen eingesetzt werden. Solche Metriken wurden zwar im Rahmen dieser Arbeit erhoben (siehe [Bis16]), aber nicht im vollen Umfang genutzt.

Tobergte et al. [TC13] sehen eine Notwendigkeit eines branchenweiten Repositorys für Beschreibungen von Abweichungen und Ibidunmoye et al. [IHRE15] sehen einen öffentlich zugänglichen Datenspeicher für Performanzszenarien als notwendig an. Die Erstellung solcher Datenbanken für Beschreibung von Änderungenin standardisierten Formaten, wie beispielsweise im CRI-Modell, wäre ebenso erstrebenswert. Dies würden neue Möglichkeiten einer empirisch geprägten Evolutionsforschung auf der Grundlage einer gemeinsamen Datenbasis eröffnen. Technisch wären diesbezügliche vorhandene Repositorys von Modellen der Entwicklung bzw. Open-Source Systemen von großem Forschungsinteresse.

Die Forschungstätigkeiten im Rahmen dieser Arbeit haben **Synergien zur modellbasierten Entwicklungsmethoden** und den dort verwendeten Entwicklungsartefakten aufgezeigt. Dabei beschreiben Entwicklungsartefakte und Evolutionsartefakte beide die Struktur oder das Verhalten von Systemteilen. Allerdings werden Entwicklungsartefakte als präskriptive Beschreibung verwendet indem sie beschreiben wie das System aufgebaut wird und Evolutionsartefakte stellen üblicherweise das momentane Verhalten dar und beschreiben das System somit deskriptiv (vergleiche [Küh06]). Dabei basieren Entwicklungsartefakte auf geschätzten und vermuteten Werten, während generierte Evolutionsartefakte gemessene Werte verwenden. Die natürliche Schlussfolgerung ist, dass beide Typen von Artefakten miteinander kombiniert werden können, indem die beabsichtigte mit der tatsächlichen Situation verglichen wird.

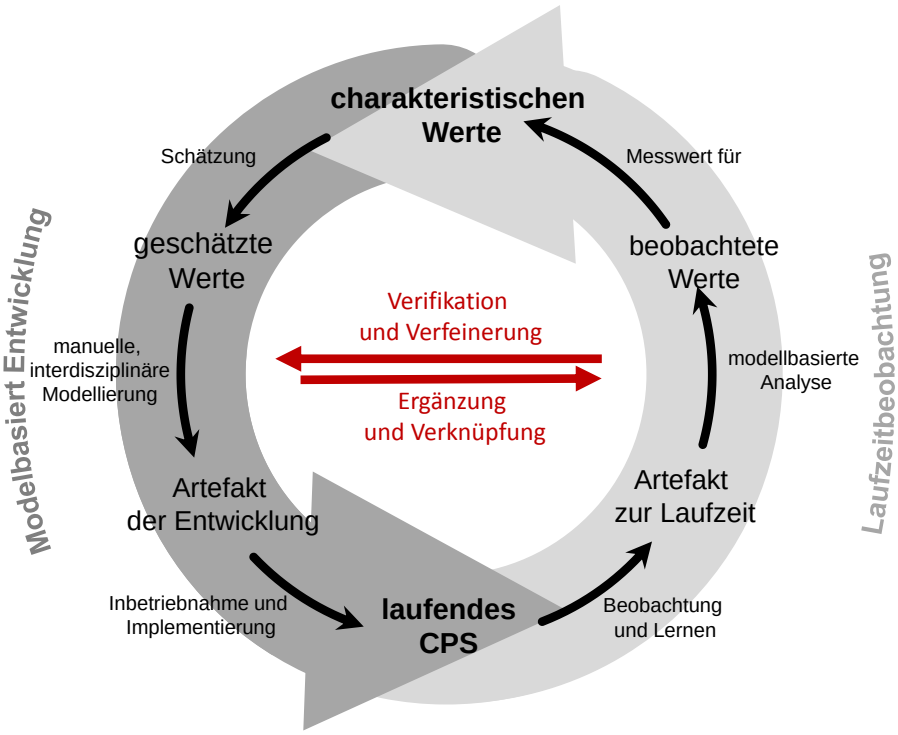


Abbildung 7.2: Wechselwirkung zwischen modellbasierter Entwicklung und Beobachtung zur Laufzeit

Auch diese Schlussfolgerung sollte in möglichen weiterführenden Forschungsarbeiten näher untersucht werden. Der zyklische Prozess aus Abbildung 7.2 zeigt dabei die im Rahmen dieser Arbeit dokumentierten Synergien (siehe [HLL+14]). Mit Entwicklungsmethoden werden dabei über charakteristische Werte Artefakte modelliert, die ein System beschreiben. Diese Entwicklungsartefakte werden dann manuell oder durch Codegenerierung implementiert und in Betrieb genommen. Durch die in dieser Arbeit vorgenommene Generierung von Modellartefak-

ten werden genau diese charakteristischen Werte sichtbar und darüber hinaus auch messbar. Dadurch ergeben sich zwei Möglichkeiten wie diese Werte verknüpft werden können: Zum einen können Informationen aus den koevolvierenden Modellartefakten genutzt werden, um die ursprünglich geschätzten Werte zu überprüfen und dadurch die Entwicklungsmodelle in den Evolutionsprozess einzubeziehen. Ein solcher Ansatz ist ein Rückkanal der in dieser Arbeit verfolgten Unterstützungsmethodik. Die zweite Möglichkeit wäre die direkte Nutzung von charakteristischen Werten der Entwicklungsmodelle für die Evolution. Dies wäre im Sinne der Arbeit eine zusätzliche Kontextquelle, wodurch bei der inhärenten Komplexität und Größe von cyber-physischen Systemen der manuelle und fehleranfällige Aufwand einer Ermittlung des Kontextes reduziert werden würde.

Die verwendeten Modelle beschreiben jeweils nur einzelne Teile eines cyber-physischen Systems, weshalb die Modelle jeweils hohe Abhängigkeiten voneinander aufweisen. Diese Abhängigkeit zwischen Modellen ist im Rahmen der vorgestellten Evolutionsunterstützung bisher nicht betrachtet worden. Diesbezüglich bietet die unterliegende Modelltransformation allerdings die Möglichkeiten einer Koevolution zwischen Modellen, wie anhand von Architektur- und Fehlermodellen gezeigt wurde (siehe [GGv+18]). Diese Erweiterung wäre insbesondere bezüglich interdisziplinärer Modelle, wie beispielsweise von Legat et al. [LMC+14] für cyber-physische Produktionssysteme vorgeschlagen, interessant. Denn über eine solche Koevolution von Modellen könnten die Abhängigkeiten der Disziplinen besser abgebildet und die Modelle somit verknüpft werden. Dies würde eine bessere interdisziplinäre Sichtweise in Vorschlägen ermöglichen und damit den Nutzer in die Lage versetzten Auswirkung zwischen den Disziplinen abzuschätzen und diese Auswirkungen zurückzuverfolgen.

Die Koevolution der Modelle wird in dieser Arbeit über einen halbautomatisierten Prozess mit einer agentenbasierten Belief-Desire-Intention-Architektur gesteuert. Eine solche Agentenarchitektur könnte auch die **Entscheidungsfindung** des Nutzers ersetzten indem ein geeigneter Mechanismus definiert wird, der Entscheidungen auf Basis der vorliegenden Informationen trifft. Dazu müsste ein Agent in die Lage versetzt werden, beispielsweise anhand von vergleichbaren Systemen zu entscheiden, ob eine Evolution gewollt oder ungewollt ist. Für eine solche Entscheidungsarchitektur gäbe es insbesondere bei erhaltender Evolution, eine Vielzahl von interessanter Anwendungsfällen: So könnte beispielsweise für weit verbreitete Systeme überprüft werden, ob Evolutionsschritte vorliegen, die vom Hersteller, wie im Fall eines durch eine aktualisierte Version gelernten Verhaltens, gewollt oder, wie im Fall eines durch Malware infizierten Gerät, ungewollt sind. Diesbezüglich wäre auch der Einsatz von Deep Learning-Verfahren für die Entscheidungsfindung denkbar.

Eine erweiterte Unterstützung des Nutzers ist auch bei den angebotenen Analysemethoden denkbar. Walter [Wal18] schlägt beispielsweise für die Analyse von Komponentensystemen eine beschreibende Sprache für performanzrelevante Fragestellungen, eine Entscheidungsunterstützung für Leistungsbewertungsansätze und eine Schätzung der Analysezeit für die modellbasierte Leistungsvorhersage vor. Weiterhin bieten auch die Simulation der Performanz, wie sie durch das Palladio Komponentenmodell angebotenen wird, Potenziale für eine umfangreichere Analyse der Modelle. Diese Analysemethoden können genauso wie die Ableitung von Vorschlägen in die Evolutionsunterstützung integriert werden. Dies gilt auch für eine zusätzliche Validierung und Verifizierung der durch die wissenstragende Komponente vorgeschlagene Handlungsalternativen. Denn dadurch könnte die Aussage über die Anwendbarkeit im realen System verbessert werden. Denn eine Aussage über die Implementierbarkeit von Vorschlägen oder sogar eine Selbstadaption ist durch den in dieser Arbeit vorgestellten Ansatz noch nicht möglich. Generell ließe sich durch zusätzliche Methodiken ein umfassenderes Bild bei der Betrachtung der abgebildeten Versionen und Varianten vermitteln.

Die im Rahmen dieser Arbeit entwickelten Evolutionsschritte sind neben der Fortführung von Evolution auch anderweitig verwendbar. Als **weiterer Verwendungszweck** können z.B. modellbasierte Evolutionsschritte genutzt werden, um über deren Reaktionen Testfälle zu erzeugen. Diese Testfälle würden es beispielsweise erlauben, den Ansatz der Arbeit mit Verfahren zur Operationsbestimmung durch Beispiele (vergleiche [BLS+09, Lan11]) zu koppelt. Dabei wäre eine Verknüpfung in beide Richtungen möglich indem nicht nur Testfälle aus Evolutionsschritten, sondern auch Testfällen als Evolutionsschritte generiert werden. Dies wurde in Rahmen einer Forschungskooperation bereits teilweise mit der Generierung von Operationen der Modelltransformation für Materialflussmodelle durch generische Algorithmen prototypisch erprobt (siehe [KGT16]).

Die implementiere Systemarchitektur dieser Arbeit ist dafür vorgesehen, u.a. über die dabei zugrundeliegenden *Aktiven Komponenten* auf den unterschiedlichen Ebenen eines cyber-physikalischen Systems weitere Erweiterungen einzubinden. Derartige Erweiterungen auf den unteren Ebenen des cyber-physikalischen Systems sind beispielsweise neue Artefakttypen oder nicht ereignisbasierte Systeme. Auf den oberen Ebenen sind dies beispielsweise Verfahren zur Verbesserung der Latenzzeiten oder eine Betrachtung von Netzwerkausfällen.

Erweiterungen könnten auch **funktionale Verbesserungen** umfassen. Dies gilt insbesondere für den in dieser Arbeit vorgeschlagenen Marktplatz, der mit Blick auf Trends wie dem Internet der Dinge eine hohe Zahl an Teilnehmern beinhalten könnte. Denn für Netzwerke mit vielen Teilnehmern ist die in dieser Arbeit nur rudimentär betrachtete Suche und Bewertung von Evolutionsschritten nicht ausreichend. An dieser Stelle wäre eine Cluster- oder Trendanalyse geeignet, um passende Evolutionsschritte in größeren Netzwerken zu identifizieren.

Weiterhin konzentriert sich die hier vorliegende Arbeit hauptsächlich auf positive Erfahrungen der Evolution von cyber-physischen Systemen. Unter realen Bedingungen können allerdings auch eine Vielzahl an negativen, ungewollten Änderungen beobachtet werden und als Erfahrungen weitergegeben werden. Denn dadurch könnte untersucht werden, ob durch den Ansatz auch schädlichen Einflüsse und Fehler in der operativen Phase identifiziert und deren Behebung systematisch unterstützt werden könnte. Dadurch könnten proaktiv fehlerspezifische Hilfestellungen angeboten werden und es könnte die typischerweise natürlichsprachliche Hilfe in Foren durch modellbasierte Hilfsangebote verbessert werden.

Ein solches Szenario wäre auch geeignet, um vertrauensbasierte Ansätze für verteilten Netzwerken zu untersuchen. Dabei könnte im Besonderen die Relation der Vererbung eines Evolutionsprozesses als Vertrauensmaß für Teilnehmer dienen. Dadurch würde die Vision eines Marktplatzes für Evolutionsschritte weiterverfolgt werden, indem Vertrauen vergleichbar zu Hilfeforen, wie *Stack Overflow*, oder zu Bewertungssystemen, wie z.B. *Google Page Ranks*, eingesetzt wird. Die prototypische Implementierung bietet mit der hier verwendeten Blockchain-Technologie schon eine dafür notwendige und passende Systemarchitektur. Aufbauend auf der bereits in dieser Arbeit verwendeten Blockchain wurden u.a. schon dezentrale Abrechnungs- und Auftragsvergabemechanismen positiv erprobt (siehe [PKBL18]). In diesem Zusammenhang könnten weiterhin auch homomorphe Verschlüsselungsverfahren (siehe [GB09]) zusätzliche Datenschutz- und Sicherheitsmechanismen liefern.

Zusammenfassend kann so das im Rahmen dieser Arbeit vorgeschlagene und untersuchte Konzept von Evolutionsschritten auch in weiteren, ganz unterschiedlichen Kontexten und Anwendungsgebieten zusätzliche Forschungsbeiträge leisten. Die in dieser Arbeit entwickelte Evolutionsunterstützung kann dabei verwandte modellbasierte Verfahren integrieren und die Systemarchitektur einer wissenstragenden Komponente für weitere Anwendungsfälle genutzt werden.

Eigene Veröffentlichungen

Folgende Veröffentlichungen des Autors sind aus den Forschungstätigkeiten zu dieser Arbeit hervorgegangen:

1. **Haubeck, Christopher**; Pokahr, Alexander; Reichert, Kim; Hohenberger, Till; Lamersdorf, Winfried (2018): The CRI-Model: A Domain-independent Taxonomy for Non-Conformance between Observed and Specified Behaviour. In: Computer Science & Information Systems, 15(3).
2. **Haubeck, Christopher**; Bornholdt, Heiko; Lamersdorf, Winfried; Chakraborty, Abhishek; Fay, Alexander (2018): Step-based Evolution Support among Networked Production Automation Systems. In: at-Automatisierungstechnik, 66(10).
3. **Haubeck, Christopher**; Wior, Ireneus; Braubach, Lars; Pokahr, Alexander; Ladiges, Jan; Fay, Alexander; Lamersdorf, Winfried (2013): Keeping Pace with Changes-Towards Supporting Continuous Improvements and Extensive Updates in Manufacturing Automation Software. In: Electronic Communications of the EASST 56.
4. **Haubeck, Christopher**; Chakraborty, Abhishek; Ladiges, Jan; Pokahr, Alexander; Lamersdorf, Winfried; Fay, Alexander (2017): Evolution of Cyber-Physical Production Systems supported by community-enabled experiences. In: IEEE 15th International Conference of Industrial Informatics (INDIN), IEEE, S. 867-874.
5. **Haubeck, Christopher**; Lamersdorf, Winfried; Ladiges, Jan; Fay, Alexander (2014): An active service-component architecture to enable self-awareness of evolving production systems. In: Emerging Technology and Factory Automation (ETFA), 2014, IEEE, S. 1–8.
6. **Haubeck, Christopher**; Lamersdorf, Winfried; Ladiges, Jan; Fay, Alexander; Fuchs, Julia; Legat, Christoph; Vogel-Heuser, Birgit (2014): Interaction of model-driven engineering and signal-based online monitoring of production systems. Towards Requirement-aware evolution. In: Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE. IEEE, S. 2571–2577.
7. **Haubeck, Christopher**; Vilenica, Ante; Lamersdorf, Winfried (2013): Using Building Blocks for Pattern-Based Simulation of Self-organising Systems. In: Intelligent Distributed Computing VI: Springer, S. 9–15.
8. **Haubeck, Christopher**; Lamersdorf, Winfried; Fay, Alexander (2018): A Knowledge Carrying Service-Component Architecture for Smart Cyber Physical Systems: An Example based on self-documenting production systems. In: ICSOC Workshops 2017, Bd. 10979 (Lecture Notes in Computer Science): Springer-Verlag, Berlin Heidelberg New York, S. 270–282.
9. **Haubeck, Christopher**; Ladiges, Jan; Lamersdorf, Winfried; Fay, Alexander (2014): Behandlung unbekannter Änderungen in automatisierten Produktionsprozessen anhand von Wissensmodellen. In: Software Engineering (Workshops), S. 2–3.

10. Chakraborty, Abhishek; **Haubeck, Christopher**; Fay, Alexander; Lamersdorf, Winfried (2018): Signal-based Context Comparative Analysis for Identification of Similar Manufacturing Modules. In: IFAC-PapersOnLine 51 (11), S. 276–283.

11. Jürjens, Jan; Schneider, Kurt; Bürger, Jens; Viertel, Fabien Patrick; Strüber, Daniel; Goedicke, Michael; Reussner, Ralf; Heinrich, Robert; Taspolatoglu, Emre; Konersmann, Marco; Fay, Alexander; Lamersdorf, Winfried; Ladiges, Jan; **Haubeck, Christopher** (2019): Maintaining Security in Software Evolution: Kapitel 2. In: DFG SPP1593 - Book of Results: Springer-Verlag, Heidelberg - New York (in Veröffentlichung).

12. Kögel, Stefan; Tichy, Matthias; Chakraborty, Abhishek; Fay, Alexander; Vogel-Heuser, Birgit; **Haubeck, Christopher**; Taentzer, Gabriele; Kehrer, Timo; Ladiges, Jan; Grunske, Lars; Bougouffa, Safa; Getir, Sinem; Cha, Suhyun; Kelter, Udo; Lamersdorf, Winfried; Busch, Kiana; Heinrich, Robert; Koch, Sandro (2018): Learning from Evolution for Evolution: Kapitel 6. In: DFG SPP1593 - Book of Results: Springer-Verlag, Heidelberg - New York (in Veröffentlichung).

13. Ladiges, Jan; **Haubeck, Christopher**; Fay, Alexander; Lamersdorf, Winfried (2014): Evolution management of production facilities by semi-automated requirement verification. In: at-Automatisierungstechnik 62 (11), S. 781–793.

14. Ladiges, Jan; Fay, Alexander; **Haubeck, Christopher**; Lamersdorf, Winfried (2013): Operationalized definitions of non-functional requirements on automated production facilities to measure evolution effects with an automation system. In: Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on. IEEE, S. 1–6.

15. Ladiges, Jan; Fay, Alexander; **Haubeck, Christopher**; Lamersdorf, Winfried; Lity, Sascha; Schaefer, Ina (2015): Supporting commissioning of production plants by model-based testing and model learning. In: Industrial Electronics (ISIE), 2015 IEEE 24th International Symposium on. IEEE, S. 606–611.

16. Ladiges, Jan; Fülber, Alexander; Arroyo, Esteban; Fay, Alexander; **Haubeck, Christopher**; Lamersdorf, Winfried (2015): Learning material flow models for manufacturing plants from data traces. In: Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on. IEEE, S. 294–301.

17. Ladiges, Jan; **Haubeck, Christopher**; Fay, Alexander; Lamersdorf, Winfried (2014): Semiautomated decision making support for undocumented evolutionary changes. In: Workshop Software-Reengineering & Evolution.

18. Ladiges, Jan; **Haubeck, Christopher**; Fay, Alexander; Lamersdorf, Winfried (2015): Learning behaviour models of discrete event production systems from observing input/output signals. In: IFAC-PapersOnLine 48 (3), S. 1565–1572.

19. Ladiges, Jan; Wior, Ireneus; Arroyo, Esteban; Fay, Alexander; **Haubeck, Christopher**; Lamersdorf, Winfried (2013): Evolution of production facilities and its impact on non-functional requirements. In: Industrial Informatics (INDIN), 2013 11th IEEE International Conference on. IEEE, S. 224–229.

20. Pietsch, Christopher; Kelter, Udo; **Haubeck, Christopher**; Lamersdorf, Winfried; Chakraborty, Abhishek; Fay, Alexander (2018): Using Model Differencing to reason about Observable Behaviour Changes of Manufacturing Systems. In: at-Automatisierungstechnik, 66(10).

21. Pokahr, Alexander; Braubach, Lars; **Haubeck, Christopher**; Ladiges, Jan (2014): Programming BDI agents with pure Java. In: German Conference on Multiagent System Technologies. Springer, S. 216–233.

22. Reichert, Kim; Pokahr, Alexander; Hohenberger, Till; **Haubeck, Christopher**; Lamersdorf, Winfried (2017): A Taxonomy of Anomalies in Distributed Cloud Systems. The CRI-Model. In: International Symposium on Intelligent and Distributed Computing. Springer, S. 247–261.

23. Vogel-Heuser, Birgit; Feldmann, Stefan; Folmer, Jens; Kowal, Matthias; Schaefer, Ina; Ladiges, Jan; Fay, Alexander; **Haubeck, Christopher**; Lamersdorf, Winfried; Lity, Sascha; Kehrer, Timo; Tichy, Matthias; Getir, Sinem; Ulbrich Matthias; Klebanov, Vladimir; Beckert, Bernhard (2015): Selected challenges of software evolution for automated production systems. In: Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on. IEEE, S. 314–321.

Literaturverzeichnis

[AAH+11] AHMADI, HOSSEIN, TAREK ABDELZAHER, JIAWEI HAN, NAM PHAM und RAGHU K. GANTI: *The sparse regression cube: A reliable modeling technique for open cyber-physical systems*. In: *Proceedings of the 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, Seiten 87–96, 2011.

[ACH+10] ACKERMANN, CHRIS, RANCE CLEAVELAND, SAMUEL HUANG, ARNAB RAY, CHARLES SHELTON und ELIZABETH LATRONICO: *Automatic requirement extraction from test cases*. In: *International Conference on Runtime Verification*, Seiten 1–15, 2010.

[AD97] APFELBAUM, LARRY und JOHN DOYLE: *Model based testing*. In: *Software Quality Week Conference*, Seiten 296–300, 1997.

[AFG+10] ARMBRUST, MICHAEL, ARMANDO FOX, REAN GRIFFITH, ANTHONY D. JOSEPH, RANDY KATZ, ANDY KONWINSKI, GUNHO LEE, DAVID PATTERSON, ARIEL RABKIN, ION STOICA und OTHERS: *A view of cloud computing*. Communications of the ACM, 53(4):50–58, 2010.

[AGJ+14] ASSMANN, UWE, SEBASTIAN GÖTZ, JEAN-MARC JÉZÉQUEL, BRICE MORIN und MARIO TRAPP: *A reference architecture and roadmap for Models@ run. time systems*. In: *Models@ run. time*, Seiten 1–18. Springer, 2014.

[AKCW15] AMBHORE, NITIN, DINESH KAMBLE, SATISH CHINCHANIKAR und VISHAL WAYAL: *Tool condition monitoring system: A review*. Materials Today: Proceedings, 2(4-5):3419–3428, 2015.

[AKT+18] ALEXANDER, STEFAN KÖGEL, MATTHIAS TICHY, ABHISHEK CHAKRABORTY, ALEXANDER FAY, BIRGIT VOGEL-HEUSER, CHRISTOPHER HAUBECK, GABRIELE TAENTZER, TIMO KEHRER, LARS GRUNSKE, MATTIAS ULBRICH, SAFA BOUGOUFFA, SINEM GETIR, SUHYUN CHA, UDO KELTER, KIANA BUSCH, ROBERT HEINRICH und SANDRO KOCH: *Learning from Evolution for Evolution: Kapitel 6*. In: *DFG SPP1593 - Book of Results*. Springer-Verlag, Heidelberg - New York, 2018.

[ALR04] AVIZIENIS, A., J.-C. LAPRIE und B. RANDELL: *Dependability and its Threats: A Taxonomy*. In: *Proc. IFIP 18th World Computer Congress*, Seiten 91–120, 2004.

[Arn96] ARNOLD, ROBERT S.: *Software change impact analysis*. IEEE Computer Society Press, 1996.

[ARS15] ARCAINI, PAOLO, ELVINIA RICCOBENE und PATRIZIA SCANDURRA: *Modeling and analyzing MAPE-K feedback loops for self-adaptation*. In: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Seiten 13–23, 2015.

[Art88] ARTHUR, LOWELL JAY: *Software evolution: The software maintenance challenge*. Wiley-Interscience, 1988.

[AWR91] AIKEN, LEONA S., STEPHEN G. WEST und RAYMOND R. RENO: *Multiple regression: Testing and interpreting interactions*. Sage, 1991.

[BBM+15] BERARDINELLI, LUCA, STEFAN BIFFL, EMANUEL MAETZLER, TANJA MAYER-

HOFER und MANUEL WIMMER: *Model-based co-evolution of production systems and their libraries with AutomationML*. In: *Conf. on Emerging Technologies and Factory Automation*, Seiten 1–8, 2015.

[BDF+11] BOSCHIAN, VALENTINA, MARIAGRAZIA DOTOLI, MARIA PIA FANTI, GIORGIO IACOBELLIS und WALTER UKOVICH: *A metamodeling approach to the management of intermodal transportation networks*. Transactions on Automation Science and Engineering, 8(3):457–469, 2011.

[BDG+09] BRUN, YURIY, GIOVANNA MARZO DI SERUGENDO, CRISTINA GACEK, HOLGER GIESE, HOLGER KIENLE, MARIN LITOIU, HAUSI MÜLLER, MAURO PEZZÈ und MARY SHAW: *Engineering self-adaptive systems through feedback loops*. In: *Software engineering for self-adaptive systems*, Seiten 48–70. Springer, 2009.

[BDIM04] BARHAM, PAUL, AUSTIN DONNELLY, REBECCA ISAACS und RICHARD MORTIER: *Using Magpie for request extraction and workload modelling*. In: *OSDI*, Band 4, Seite 18, 2004.

[BE09] BENDIX, LARS und PAR EMANUELSSON: *Collaborative work with software models-industrial experience and requirements*. In: *Model-Based Systems Engineering, 2009. MBSE'09. International Conference on*, Seiten 60–68, 2009.

[BFH+10] BROY, MANFRED, MARTIN FEILKAS, MARKUS HERRMANNSDOERFER, STEFANO MERENDA und DANIEL RATIU: *Seamless model-based development: From isolated tools to integrated model engineering environments*. Proceedings of the IEEE, 98(4):526–545, 2010.

[BG09] BERNS, ANDREW und SUKUMAR GHOSH: *Dissecting self-* properties*. In: *Self-Adaptive and Self-Organizing Systems, 2009. SASO'09. Third IEEE International Conference on*, Seiten 10–19, 2009.

[BHH15] BIELEFELDT, BRENT, JACOB HOCHHALTER und DARREN HARTL: *Computationally efficient analysis of SMA sensory particles embedded in complex aerostructures using a substructure approach*. In: *ASME 2015 Conference on Smart Materials, Adaptive Structures and Intelligent Systems*, 2015.

[Bin07] BINKLEY, DAVID: *Source code analysis: A road map*. In: *2007 Future of Software Engineering*, Seiten 104–119, 2007.

[Bis16] BISCHOF, PAUL: *Metrische Evaluation der Zukunftsfähigkeit von Webframeworks*. Bachelorarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, 2016.

[BK13] BETTENHAUSEN, KURT D. und STEFAN KOWALEWSKI: *Cyber-physical systems: Chancen und Nutzen aus Sicht der Automation*. VDI/VDE-Gesellschaft Mess-und Automatisierungstechnik, 2013.

[BKBR12] BROSCH, FRANZ, HEIKO KOZIOLEK, BARBORA BUHNOVA und RALF REUSSNER: *Architecture-based reliability prediction with the palladio component model*. IEEE Transactions on Software Engineering, 38(6):1319–1339, 2012.

[BKK09] BROSIG, FABIAN, SAMUEL KOUNEV und KLAUS KROGMANN: *Automated extraction of palladio component models from running enterprise Java applications*. In: *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*, Seite 10, 2009.

[BKR09] BECKER, STEFFEN, HEIKO KOZIOLEK und RALF REUSSNER: *The Palladio component model for model-driven performance prediction*. Journal of Systems and Software, 82(1):3–22, 2009.

[BLS+09] BROSCH, PETRA, PHILIP LANGER, MARTINA SEIDL, KONRAD WIELAND, MANUEL WIMMER, GERTI KAPPEL, WERNER RETSCHITZEGGER und WIELAND SCHWINGER: *An example is worth a thousand words: Composite operation modeling by-example*. In: *International Conference on Model Driven Engineering Languages and Systems*, Seiten 271–285, 2009.

[BM16] BAHGA, ARSHDEEP und VIJAY K. MADISETTI: *Blockchain platform for industrial internet of things*. Journal of Software Engineering and Applications, 9(10):533, 2016.

[BMZ+05] BUCKLEY, JIM, TOM MENS, MATTHIAS ZENGER, AWAIS RASHID und GÜNTER KNIESEL: *Towards a taxonomy of software change*. Journal of Software: Evolution and Process, 17(5):309–332, 2005.

[BP11] BRAUBACH, LARS und ALEXANDER POKAHR: *Addressing challenges of distributed systems using active components*. In: *Intelligent Distributed Computing V*, Seiten 141–151. Springer, 2011.

[BR00] BENNETT, KEITH H. und VÁCLAV T. RAJLICH: *Software maintenance and evolution: A roadmap*. In: *Proceedings of the Conference on the Future of Software Engineering*, Seiten 73–87, 2000.

[BRTD16] BANGEMANN, THOMAS, MATTHIAS RIEDL, MARIO THRON und CHRISTIAN DIEDRICH: *Integration of classical components into industrial cyber–physical systems*. Proceedings of the IEEE, 104(5):947–959, 2016.

[BS09] BELLGRAN, MONICA und EVA KRISTINA SÄFSTEN: *Production development: design and operation of production systems*. Springer Science & Business Media, 2009.

[BS12] BROY, MANFRED und KETIL STØLEN: *Specification and development of interactive systems: focus on streams, interfaces, and refinement*. Springer Science & Business Media, 2012.

[BSR04] BATORY, DON, JACOB NEAL SARVELA und AXEL RAUSCHMAYER: *Scaling stepwise refinement*. IEEE Transactions on Software Engineering, 30(6):355–371, 2004.

[BSW+09] BROSCH, PETRA, MARTINA SEIDL, KONRAD WIELAND, MANUEL WIMMER und PHILIP LANGER: *The operation recorder: specifying model refactorings by-example*. In: *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, Seiten 791–792, 2009.

[BT75] BASIL, VICTOR R. und ALBERT J. TURNER: *Iterative enhancement: A practical technique for software development*. IEEE Transactions on Software Engineering, 4:390–396, 1975.

[Buc16] BUCHMAN, ETHAN: *Tendermint: Byzantine fault tolerance in the age of blockchains*. Doktorarbeit, University of Guelph, 2016.

[BV17] BUZHINSKY, IGOR und VALERIY VYATKIN: *Automatic inference of finite-state plant models from traces and temporal properties*. Transactions on Industrial Informatics, 13(4):1521–1530, 2017.

[CBK09] CHANDOLA, V., A. BANERJEE und V. KUMAR: *Anomaly detection: A Survey*. ACM Computing Surveys, 41(3):1–15, 2009.

[CCBJ11] CÂNDIDO, GONÇALO, ARMANDO W. COLOMBO, JOSÉ BARATA und FRANÇOIS JAMMES: *Service-oriented infrastructure to support the deployment of evolvable production systems*. IEEE Transactions on Industrial Informatics, 7(4):759–767,

2011.

[CDEV08] CANFORA, GERARDO, MASSIMILIANO DI PENTA, RAFFAELE ESPOSITO und MARIA LUISA VILLANI: *A framework for QoS-aware binding and re-binding of composite web services*. Journal of Systems and Software, 81(10):1754–1769, 2008.

[CDK05] COULOURIS, GEORGE F., JEAN DOLLIMORE und TIM KINDBERG: *Distributed systems: Concepts and design*. pearson education, 2005.

[CdLL+14] CÁMARA, JAVIER, ROGÉRIO DE LEMOS, NUNO LARANJEIRO, RAFAEL VENTURA und MARCO VIEIRA: *Robustness evaluation of the rainbow framework for self-adaptation*. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, Seiten 376–383, 2014.

[CDP09] CICCHETTI, ANTONIO, DAVIDE DI RUSCIO und ALFONSO PIERANTONIO: *Model patches in model-driven engineering*. In: *Int. Conf. on Model Driven Engineering Languages and Systems*, Seiten 190–204, 2009.

[CHFL18] CHAKRABORTY, ABHISHEK, CHRISTOPHER HAUBECK, ALEXANDER FAY und WINFRIED LAMERSDORF: *Signal-based Context Comparative Analysis for Identification of Similar Manufacturing Modules*. IFAC-PapersOnLine, 51(11):276–283, 2018.

[CHK+01] CHAPIN, NED, JOANNE E. HALE, KHALED MD KHAN, JUAN F. RAMIL und WUI-GEE TAN: *Types of software evolution and software maintenance*. Journal of Software: Evolution and Process, 13(1):3–30, 2001.

[CIB+14] CAMPETELLI, ALARICO, MAXIMILIAN IRLBECK, DENIS BYTSCHKOW, M. GENGARLE und KONSTANTIN SCHORP: *Reference Framework for the Engineering of Cyber–Physical Systems: A first Approach*, 2014.

[CK94] CHIDAMBER, SHYAM R. und CHRIS F. KEMERER: *A metrics suite for object oriented design*. IEEE Transactions on Software Engineering, 20(6):476–493, 1994.

[CLRS13] CORMEN, THOMAS H., CHARLES E. LEISERSON, RONALD RIVEST und CLIFFORD STEIN: *Algorithmen-Eine Einführung*. Walter de Gruyter GmbH & Co KG, 2013.

[CMZ09] CHEN, SHIHYEN, BIN MA und KAIZHONG ZHANG: *On the similarity metric and the distance metric*. Theoretical Computer Science, 410(24-25):2365–2376, 2009.

[CRF03] COHEN, WILLIAM, PRADEEP RAVIKUMAR und STEPHEN FIENBERG: *A comparison of string metrics for matching names and records*. In: *Kdd workshop on data cleaning and object consolidation*, Band 3, Seiten 73–78, 2003.

[CvA07] CIRACI, SELIM, P. M. VAN DEN BROEK und M. AKŞIT: *A taxonomy for a constructive approach to software evolution*. Journal of software, 2(2), 2007.

[CW98a] CONRADI, REIDAR und BERNHARD WESTFECHTEL: *Version models for software configuration management*. ACM Computing Surveys (CSUR), 30(2):232–282, 1998.

[CW98b] COOK, JONATHAN E. und ALEXANDER L. WOLF: *Discovering models of software processes from event-based data*. ACM Transactions on Software Engineering and Methodology (TOSEM), 7(3):215–249, 1998.

[CWU+18] CHA, SUHYUN, ALEXANDER WEIGL, MATTIAS ULBRICH, BERNHARD BECKERT und BIRGIT VOGEL-HEUSER: *Achieving delta description of the control software for an automated production system evolution*. In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, Seiten 1170–1176, 2018.

[Dad17] DADSSI, NOUR-EDDINE: *Generierung von Palladio Performancemodellen in iterativen Entwicklungsprozessen*. Masterarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, 2017.

[Dav17] DAVIS, NICOLA: *The selfish gene*. Macat Library, 2017.

[Daw06] DAWKINS, RICHARD: *The selfish gene: with a new introduction by the author*. UK: Oxford University Press.(Originally published in 1976), 2006.

[DC14] DUMITRACHE, IOAN und SIMONA IULIANA CARAMIHAI: *Intelligent Cyber-Enterprise in the Production Context*. IFAC Proceedings Volumes, 47(3):821–826, 2014.

[DD06] DEZA, MICHEL-MARIE und ELENA DEZA: *Dictionary of distances*. Elsevier, 2006.

[DFRG08] DI MARZO SERUGENDO, GIOVANNA, JOHN FITZGERALD, ALEXANDER ROMANOVSKY und NICOLAS GUELFI: *A generic framework for the engineering of self-adaptive and self-organising systems*. In: *Dagstuhl Seminar Proceedings*, 2008.

[DG92] DALLERY, YVES und STANLEY B. GERSHWIN: *Manufacturing flow line systems: a review of models and analytical results*. Queueing systems, 12(1-2):3–94, 1992.

[DGLP08] D'AMBROS, MARCO, HARALD GALL, MICHELE LANZA und MARTIN PINZGER: *Analysing software repositories to understand software evolution*. In: *Software evolution*, Seiten 37–67. Springer, 2008.

[DIN09] DIN IEC 61131-3: *Programmable Controllers – Part 3: Programming Languages*, 2009.

[dLGM+13] LEMOS, ROGÉRIO DE, HOLGER GIESE, HAUSI A. MÜLLER, MARY SHAW, JESPER ANDERSSON, MARIN LITOIU, BRADLEY SCHMERL, GABRIEL TAMURA, NORHA M. VILLEGAS, THOMAS VOGEL und OTHERS: *Software engineering for self-adaptive systems: A second research roadmap*. In: *Software Engineering for Self-Adaptive Systems II*, Seiten 1–32. Springer, 2013.

[DLV12] DERLER, PATRICIA, EDWARD A. LEE und ALBERTO SANGIOVANNI VINCENTELLI: *Modeling cyber–physical systems*. Proceedings of the IEEE, 100(1):13–28, 2012.

[Dow97] DOWSON, MARK: *The Ariane 5 software failure*. ACM SIGSOFT Software Engineering Notes, 22(2):84, 1997.

[DP09] DUCASSE, STÉPHANE und DAMIEN POLLET: *Software architecture reconstruction: A process-oriented taxonomy*. IEEE Transactions on Software Engineering, 35(4):573–591, 2009.

[DR13] DURDIK, ZOYA und RALF REUSSNER: *On the appropriate rationale for using design patterns and pattern documentation*. In: *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, Seiten 107–116, 2013.

[dSB12] SILVA, LAKSHITHA DE und DHARINI BALASUBRAMANIAM: *Controlling software architecture erosion: A survey*. Journal of Systems and Software, 85(1):132–151, 2012.

[dSQTR07] SOUZA, CLEIDSON R. DE, STEPHEN QUIRK, ERIK TRAINER und DAVID F. REDMILES: *Supporting collaborative software development through the visualization of socio-technical dependencies*. In: *Proceedings of the 2007 international ACM conference on Supporting group work*, Seiten 147–156, 2007.

[E+07] ESTEFAN, JEFF A. und OTHERS: *Survey of model-based systems engineering (MBSE) methodologies*. Incose MBSE Focus Group, 25(8):1–12, 2007.

[EB10] EBERT, JÜRGEN und DANIEL BILDHAUER: *Reverse engineering using graph*

queries. In: *Graph transformations and model-driven engineering*, Seiten 335–362. Springer, 2010.

[ED07] EBERT, CHRISTOF und REINER DUMKE: *Software Measurement: Establish-Extract-Evaluate-Execute*. Springer Science & Business Media, 2007.

[EKS09] EBERT, JURGEN, UDO KELTER und TARJA SYSTA: *Workshop on comparison and versioning of software models (CVSM 2009)*. In: *Proceedings of the 2009 31st International Conference on Software Engineering: Companion Volume*, Seiten 457–458, 2009.

[ELM+12] EIDSON, JOHN C., EDWARD A. LEE, SLOBODAN MATIC, SANJIT A. SESHIA und JIA ZOU: *Distributed real-time software for cyber–physical systems*. Proceedings of the IEEE, 100(1):45–59, 2012.

[Erl09] ERL, THOMAS: *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2009.

[ERW08] EBERT, JÜRGEN, VOLKER RIEDIGER und ANDREAS WINTER: *Graph Technology in Reverse Engineering–The TGraph Approach*. In: *Proc. 10th Workshop Software Reengineering. GI Lecture Notes in Informatics*, 2008.

[ES07] EVANS, ROBERT B. und ALBERTO SAVOIA: *Differential testing: a new approach to change detection*. In: *The 6th Joint Meeting on European software engineering conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers*, Seiten 549–552, 2007.

[FBD15] FREITAG, MICHAEL, TILL BECKER und NEIL A. DUFFIE: *Dynamics of resource sharing in production networks*. CIRP Annals, 64(1):435–438, 2015.

[FFVH+12] FELDMANN, STEFAN, JULIA FUCHS, BIRGIT VOGEL-HEUSER und OTHERS: *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*, Seiten 1689–1698, 2012.

[FG06] FLURI, BEAT und HARALD C. GALL: *Classifying change types for qualifying change couplings*. In: *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on*, Seiten 35–45, 2006.

[FH11] FREY, SÖREN und WILHELM HASSELBRING: *The cloudmig approach: Model-based migration of software systems to cloud-optimized applications*. International Journal on Advances in Software, 4(3 and 4):342–353, 2011.

[FKF15] FLEISCHMANN, HANS, JOHANNES KOHL und JÖRG FRANKE: *A reference architecture for the development of socio-cyber-physical condition monitoring systems*. In: *Proceedings of ECIS 2015*, Seiten 1–6, 2015.

[FKVH14] FELDMANN, STEFAN, KONSTANTIN KERNSCHMIDT und BIRGIT VOGEL-HEUSER: *Combining a SysML-based modeling approach and semantic technologies for analyzing change influences in manufacturing plant models*. Procedia Cirp, 17:451–456, 2014.

[FL00] FREY, GEORG und LOTHAR LITZ: *Formal methods in PLC programming*. In: *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, Band 4, Seiten 2431–2436, 2000.

[FMNS97] FRÖHLICH, PETER, IARA MÓRA, WOLFGANG NEJDL und MICHAEL SCHRÖDER: *Diagnostic agents for distributed systems*. In: *Formal Models of Agents*, Seiten 173–186. Springer, 1997.

[FRH15] FITTKAU, FLORIAN, SASCHA ROTH und WILHELM HASSELBRING: *ExplorViz:*

Visual runtime behavior analysis of enterprise application landscapes. In: *Proceedings of ECIS 2015*, 2015.

[FS14] FITZGERALD, BRIAN und KLAAS-JAN STOL: *Continuous software engineering and beyond: Trends and challenges*. In: *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, Seiten 1–9, 2014.

[FZ97] FRANKLIN, MICHAEL und STANLEY ZDONIK: *A framework for scalable dissemination-based systems*. In: *ACM SIGPLAN Notices*, Band 32, Seiten 94–105, 1997.

[GB09] GENTRY, CRAIG und DAN BONEH: *A fully homomorphic encryption scheme*, Band 20. Stanford University Stanford, 2009.

[GB12] GEISBERGER, EVA und MANFRED BROY: *Integrierte Forschungsagenda Cyber-Physical Systems, acatech Studie*. München: TU, Institut für Informatik, 2012.

[GCH+04] GARLAN, DAVID, S-W CHENG, A-C HUANG, BRADLEY SCHMERL und PETER STEENKISTE: *Rainbow: Architecture-based self-adaptation with reusable infrastructure*. Computer, 37(10):46–54, 2004.

[Gen12] GENTLE, ANNE: *Conversation and community: The social web for documentation*. Xml Press, 2012.

[GG08] GODFREY, MICHAEL W. und DANIEL M. GERMAN: *The past, present, and future of software evolution*. In: *Frontiers of Software Maintenance*, Seiten 129–138, 2008.

[GGv+18] GETIR, SINEM, LARS GRUNSKE, ANDRÉ VAN HOORN, TIMO KEHRER, YANNIC NOLLER und MATTHIAS TICHY: *Supporting semi-automatic co-evolution of architecture and fault tree models*. Journal of Systems and Software, 142:115–135, 2018.

[GJHV11] GAMMA, ERICH, RALPH JOHNSON, RICHARD HELM und JOHN VLISSIDES: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Pearson Deutschland GmbH, 2011.

[GJL10] GRINCHTEIN, OLGA, BENGT JONSSON und MARTIN LEUCKER: *Learning of event-recording automata*. Theoretical Computer Science, 2010.

[GRG+15] GOLTZ, URSULA, RALF H. REUSSNER, MICHAEL GOEDICKE, WILHELM HASSELBRING, LUKAS MÄRTIN und BIRGIT VOGEL-HEUSER: *Design for future: Managed software evolution*. Computer Science-Research and Development, 30(3-4):321–331, 2015.

[Gro07] GROOVER, MIKELL P.: *Automation, production systems, and computer-integrated manufacturing*. Prentice Hall Press, 2007.

[HA05] HUSELIUS, JOEL und JOHAN ANDERSSON: *Model synthesis for real-time systems*. In: *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on*, Seiten 52–60, 2005.

[Har15] HART, LAURA E.: *Introduction to model-based system engineering (MBSE) and sysml*. In: *Delaware Valley INCOSE Chapter Meeting, Ramblewood Country Club, Mount Laurel, New Jersey*, 2015.

[Hat07] HATTON, LES: *How accurately do engineers predict software maintenance tasks?* Computer, 40(2), 2007.

[HBL+18] HAUBECK, CHRISTOPHER, HEIKO BORNHOLDT, WINFRIED LAMERSDORF, ABHISHEK CHAKRABORTY und ALEXANDER FAY: *Step-based Evolution Support among Networked Production Automation Systems*. at-Automatisierungstechnik,

2018.

[HBS73] HEWITT, CARL, PETER BISHOP und RICHARD STEIGER: *Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence*. In: *Advance Papers of the Conference*, Band 3, Seite 235, 1973.

[HCL+17] HAUBECK, CHRISTOPHER, ABHISHEK CHAKRABORTY, JAN LADIGES, ALEXANDER POKAHR, WINFRIED LAMERSDORF und ALEXANDER FAY: *Evolution of Cyber-Physical Production Systems supported by community-enabled experiences*. In: *IEEE 15th International Conference of Industrial Informatics INDIN 2017*, 2017.

[HD00] HOXMEIER, JOHN A. und CHRIS DICESARE: *System response time and user satisfaction: An experimental study of browser-based applications*. AMCIS 2000 Proceedings, Seite 347, 2000.

[Hei16] HEINRICH, ROBERT: *Architectural run-time models for performance and privacy analysis in dynamic cloud applications*. ACM SIGMETRICS Performance Evaluation Review, 43(4):13–22, 2016.

[HGH+15] HEINRICH, ROBERT, STEFAN GÄRTNER, TOM-MICHAEL HESSE, THOMAS RUHROTH, RALF H. REUSSNER, KURT SCHNEIDER, BARBARA PAECH und JAN JÜRJENS: *A Platform for Empirical Research on Information System Evolution*. In: *SEKE*, Seiten 415–420, 2015.

[HJZ+17] HEINRICH, ROBERT, REINER JUNG, CHRISTIAN ZIRKELBACH, WILHELM HASSELBRING und RALF REUSSNER: *An Architectural Model-Based Approach to Quality-Aware DevOps in Cloud Applicationsc*. In: *Software Architecture for Big Data and the Cloud*, Seiten 69–89. Elsevier, 2017.

[HK13] HÄSTBACKA, DAVID und SEPPO KUIKKA: *Semantics enhanced engineering and model reasoning for control application development*. Multimedia tools and applications, 65(1):47–62, 2013.

[HKVH+13] HAMETNER, R., B. KORMANN, B. VOGEL-HEUSER, D. WINKLER und A. ZOITL: *Automated Test Case Generation for Industrial Control Applications*. In: *Recent Advances in Robotics and Automation*, Seiten 263–273. Springer, 2013.

[HKW+08] HEROLD, SEBASTIAN, HOLGER KLUS, YANNICK WELSCH, CONSTANZE DEITERS, ANDREAS RAUSCH, RALF REUSSNER, KLAUS KROGMANN, HEIKO KOZIOLEK, RAFFAELA MIRANDOLA, BENJAMIN HUMMEL und OTHERS: *CoCoME-the common component modeling example*. In: *The Common Component Modeling Example*, Seiten 16–53. Springer, 2008.

[HLF18] HAUBECK, CHRISTOPHER, WINFRIED LAMERSDORF und ALEXANDER FAY: *A Knowledge Carrying Service-Component Architecture for Smart Cyber Physical Systems: An Example based on self-documenting production systems*. In: L. BRAUBACH, J.M. MURILLO, N. KAVIANI, M. LAMA, L. BURGUENO, N. MOHA, M. ORIOL (Herausgeber): *ICSOC Workshops 2017*, Band 10979 der Reihe *Lecture Notes in Computer Science*, Seiten 270–282. Springer-Verlag, Berlin Heidelberg New York, 2018.

[HLL+14] HAUBECK, CHRISTOPHER, WINFRIED LAMERSDORF, JAN LADIGES, ALEXANDER FAY, JULIA FUCHS, CHRISTOPH LEGAT und BIRGIT VOGEL-HEUSER: *Interaction of model-driven engineering and signal-based online monitoring of production systems: Towards Requirement-aware evolution*. In: *Industrial Electronics Society, IECON 2014-40th Annual Conference of the IEEE*, Seiten 2571–2577, 2014.

[HLLF14a] HAUBECK, CHRISTOPHER, JAN LADIGES, WINFRIED LAMERSDORF und ALEXANDER FAY: *Behandlung unbekannter Änderungen in automatisierten Produktionsprozessen anhand von Wissensmodellen*. In: *Software Engineering (Workshops)*, Seiten 2–3, 2014.

[HLLF14b] HAUBECK, CHRISTOPHER, WINFRIED LAMERSDORF, JAN LADIGES und ALEXANDER FAY: *An active service-component architecture to enable self-awareness of evolving production systems*. In: *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, Seiten 1–8, 2014.

[HMN15] HANSEN, HANS ROBERT, JAN MENDLING und GUSTAF NEUMANN: *Wirtschaftsinformatik*. Walter de Gruyter GmbH & Co KG, 2015.

[HMY06] HUANG, GANG, HONG MEI und FU-QING YANG: *Runtime recovery and manipulation of software architecture of component-based systems*. Automated Software Engineering, 13(2):257–281, 2006.

[HNS03] HUNGAR, HARDI, OLIVER NIESE und BERNHARD STEFFEN: *Domain-specific optimization in automata learning*. In: *International Conference on Computer Aided Verification*, Seiten 315–327, 2003.

[Hof13] HOFFMANN, DIRK W.: *Software-Qualität*. Springer-Verlag, 2013.

[HPK12] HOANG, DAT DAC, HYE-YOUNG PAIK und CHAE-KYU KIM: *Service-oriented middleware architectures for cyber-physical systems*. International Journal of Computer Science and Network Security, 12(1):79–87, 2012.

[HPR+18] HAUBECK, CHRISTOPHER, ALEXANDER POKAHR, KIM REICHERT, TILL HOHENBERGER und WINFRIED LAMERSDORF: *The CRI-Model: A Domain-independent Taxonomy for 2 Non-Conformance between Observed and Specified Behaviour*. Computer Science & Information Systems, 15(3), 2018.

[HS05] HUHNS, MICHAEL N. und MUNINDAR P. SINGH: *Service-oriented computing: Key concepts and principles*. IEEE Internet computing, 9(1):75–81, 2005.

[HVL13] HAUBECK, CHRISTOPHER, ANTE VILENICA und WINFRIED LAMERSDORF: *Using Building Blocks for Pattern-Based Simulation of Self-organising Systems*. In: *Intelligent Distributed Computing VI*, Seiten 9–15. Springer, 2013.

[HWB+13] HAUBECK, CHRISTOPHER, IRENEUS WIOR, LARS BRAUBACH, ALEXANDER POKAHR, JAN LADIGES, ALEXANDER FAY und WINFRIED LAMERSDORF: *Keeping Pace with Changes-Towards Supporting Continuous Improvements and Extensive Updates in Manufacturing Automation Software*. Electronic Communications of the EASST, 56, 2013.

[HWÖ+10] HAMETNER, R., D. WINKLER, T. ÖSTREICHER, S. BIFFL und A. ZOITL: *The Adaptation of Test-Driven Software Processes to Industrial Automation Engineering*. In: *INDIN'2010*, Seiten 921–927, 2010.

[ICH11] IVANOVIĆ, DRAGAN, MANUEL CARRO und MANUEL HERMENEGILDO: *Constraint-based runtime prediction of SLA violations in service orchestrations*. In: *International Conference on Service-Oriented Computing*, Seiten 62–76, 2011.

[IHRE15] IBIDUNMOYE, OLUMUYIWA, FRANCISCO HERNÁNDEZ-RODRIGUEZ und ERIK ELMROTH: *Performance anomaly detection and bottleneck identification*. ACM Computing Surveys (CSUR), 48(1):4, 2015.

[Int] INTERNATIONAL ELECTROTECHNICAL COMMISSION: *International Electrotechnical Vocabulary*.

[Int95] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO/IEC 12207, Sy-*

stems and software engineering - Software life cycle processes, 1995.

[Int05] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION: *ISO/IEC 25000, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE)*, 2005.

[Ise06] ISERMANN, ROLF: *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2006.

[IWF07] ISRAR, TAUSEEF, MURRAY WOODSIDE und GREG FRANKS: *Interaction tree algorithms to extract effective architecture and layered performance models from traces*. Journal of Systems and Software, 80(4):474–492, 2007.

[Jaz05] JAZAYERI, MEHDI: *Species evolve, individuals age*. In: *Principles of Software Evolution, Eighth International Workshop on*, Seiten 3–9, 2005.

[JB05] JANSEN, ANTON und JAN BOSCH: *Software architecture as a set of architectural design decisions*. In: *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on*, Seiten 109–120, 2005.

[JBNK14] JHA, P. C., VIKRAM BALI, SONAM NARULA und MALA KALRA: *Optimal component selection based on cohesion & coupling for component based software system under build-or-buy scheme*. Journal of Computational Science, 5(2):233–242, 2014.

[JCL11] JENSEN, JEFF C., DANICA H. CHANG und EDWARD A. LEE: *A model-based design methodology for cyber-physical systems*. In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, Seiten 1666–1671, 2011.

[JDL16] JIANG, PINGYU, KAI DING und JIEWU LENG: *Towards a cyber-physical-social-connected and service-oriented manufacturing paradigm: Social Manufacturing*. Manufacturing Letters, 7:15–21, 2016.

[Jen11] JENKINS, JON: *Velocity Culture*, 2011.

[JFWL11] JÄGER, TOBIAS, ALEXANDER FAY, THOMAS WAGNER und ULRICH LÖWEN: *Mining technical dependencies throughout engineering process knowledge*. In: *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, Seiten 1–7, 2011.

[JHS13] JUNG, REINER, ROBERT HEINRICH und ERIC SCHMIEDERS: *Model-driven instrumentation with Kieker and Palladio to forecast dynamic applications*. In: *Symposium on Software Performance: Joint Kieker/Palladio Days 2013*, Band 1083, Seiten 99–108, 2013.

[JMK+13] JUNG, GUEYOUNG, TRIDIB MUKHERJEE, SHRUTI KUNDE, HYUNJOO KIM, NAVEEN SHARMA und FRANK GOETZ: *Cloudadvisor: A recommendation-as-a-service platform for cloud configuration and pricing*. In: *2013 IEEE Ninth World Congress on Services*, Seiten 456–463, 2013.

[Jon10] JONES, DON: *The Five Essential Elements of Application Performance Monitoring*. Quest Software. Nov, Seite 2, 2010.

[JOX10] JIN, WEI, ALESSANDRO ORSO und TAO XIE: *Automated behavioral regression testing*. In: *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, Seiten 137–146, 2010.

[JSB+18] JÜRJENS, JAN, KURT SCHNEIDER, JENS BÜRGER, FABIEN PATRICK VIERTEL, DANIEL STRÜBER, MICHAEL GOEDICKE, RALF REUSSNER, ROBERT HEINRICH, EMRE TASPOLATOGLU, MARCO KONERSMANN, ALEXANDER FAY, WINFRIED LAMERSDORF, JAN LADIGES und CHRISTOPHER HAUBECK: *Maintaining Security in Software Evolution: Kapitel 2*. In: *DFG SPP1593 - Book of Results*. Springer-

Verlag, Heidelberg - New York, 2018.

[Jun16] JUNKER, MAXIMILIAN: *Specification and Analysis of Availability for Software-Intensive Systems*. Doktorarbeit, Technische Universität München, 2016.

[KB15] KÜHNLE, HERMANN und GÜNTER BITSCH: *Foundations & principles of distributed manufacturing*. Springer, 2015.

[KDPP09] KOLOVOS, DIMITRIOS S., DAVIDE DI RUSCIO, ALFONSO PIERANTONIO und RICHARD F. PAIGE: *Different models for model matching: An analysis of approaches to support model differencing*. In: *ICSE Workshop on Comparison and Versioning of Software Models (CVSM)*, Seiten 1–6, 2009.

[Keh15] KEHRER, TIMO: *Calculation and propagation of model changes based on user-level edit operations: A foundation for version and variant management in model-driven engineering*. Doktorarbeit, Universität Siegen, 2015.

[Kel10] KELTER, UDO: *Pseudo-Modelldifferenzen und die Phasenabhängigkeit von Metamodellen*. In: *Software Engineering*, Seiten 117–128, 2010.

[KET+17] KLEMS, MARKUS, JACOB EBERHARDT, STEFAN TAI, STEFFEN HÄRTLEIN, SIMON BUCHHOLZ und AHMED TIDJANI: *Trustless intermediation in blockchain-based decentralized service marketplaces*. In: *International Conference on Service-Oriented Computing*, Seiten 731–739, 2017.

[KGT16] KÖGEL, STEFAN, RAFFAELA GRONER und MATTHIAS TICHY: *Automatic Change Recommendation of Models and Meta Models Based on Change Histories*. In: *ME@ MODELS*, Seiten 14–19, 2016.

[KHF+16] KWON, DAEIL, MELINDA R. HODKIEWICZ, JIAJIE FAN, TADAHIRO SHIBUTANI und MICHAEL G. PECHT: *IoT-based prognostics and systems health management for industrial applications*. IEEE Access, 4:3659–3670, 2016.

[KHMD12] KRAUSE, JAN, ELKE HINTZE, STEPHAN MAGNUS und CHRISTIAN DIEDRICH: *Model based specification, verification, and test generation for a safety fieldbus profile*. In: *International Conference on Computer Safety, Reliability, and Security*, Seiten 87–98, 2012.

[KHV06] KOTHAMASU, RANGANATH, SAMUEL H. HUANG und WILLIAM H. VERDUIN: *System health monitoring and prognostics—a review of current paradigms and practices*. The International Journal of Advanced Manufacturing Technology, 28(9-10):1012–1024, 2006.

[KIT93] KAO, W-I, RAVISHANKAR K. IYER und DONG TANG: *FINE: A fault injection and monitoring environment for tracing the UNIX system behavior under faults*. IEEE Transactions on Software Engineering, 19(11):1105–1118, 1993.

[KKK13] KELTER, UDO, TIMO KEHRER und DENNIS KOCH: *Patchen von Modellen*. In: *Software Engineering*, Seiten 171–184, 2013.

[KKOS12] KEHRER, TIMO, UDO KELTER, MANUEL OHRNDORF und TIM SOLLBACH: *Understanding model evolution through semantically lifting model differences with SiLift*. In: *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, Seiten 638–641, 2012.

[KKPS12] KEHRER, TIMO, UDO KELTER, PIT PIETSCH und MAIK SCHMIDT: *Adaptability of model comparison tools*. In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, Seiten 306–309, 2012.

[KKT11] KEHRER, TIMO, UDO KELTER und GABRIELE TAENTZER: *A rule-based approach to the semantic lifting of model differences in the context of model versioning*. In:

Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, Seiten 163–172, 2011.

[KKT13] KEHRER, TIMO, UDO KELTER und GABRIELE TAENTZER: *Consistency-preserving edit scripts in model versioning*. In: *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, Seiten 191–201, 2013.

[KKT14] KEHRER, TIMO, UDO KELTER und GABRIELE TAENTZER: *Propagation of software model changes in the context of industrial plant automation*. at-Automatisierungstechnik, 62(11):803–814, 2014.

[KLM03] KAISER, BERNHARD, PETER LIGGESMEYER und OLIVER MÄCKEL: *A new component concept for fault trees*. In: *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33*, Seiten 37–46, 2003.

[Kön10] KÖNEMANN, PATRICK: *Capturing the intention of model changes*. In: *Int. Conf. on Model Driven Engineering Languages and Systems*, Seiten 108–122, 2010.

[Kri88] KRIZ, JÜRGEN: *Facts and artefacts in social science: An epistemological and methodological analysis of empirical social science research techniques*. McGraw-Hill, 1988.

[Kro12] KROGMANN, KLAUS: *Reconstruction of software component architectures and behaviour models using static and dynamic analysis*, Band 4. KIT Scientific Publishing, 2012.

[Küh06] KÜHNE, THOMAS: *Matters of (meta-) modeling*. Software & Systems Modeling, 5(4):369–385, 2006.

[LAD+11] LIN, SHANG-WEI, ÉTIENNE ANDRÉ, JIN SONG DONG, JUN SUN und YANG LIU: *An efficient algorithm for learning event-recording automata*. In: *International Symposium on Automated Technology for Verification and Analysis*, Seiten 463–472, 2011.

[Lad18] LADIGES, JAN: *Automatisierte Bestimmung von Eigenschaften industrieller Produktionssysteme unter Einfluss evolutionärer Änderungen*. Dissertation, Helmut Schmidt Universität, Holstenhofweg 85, 22043 Hamburg, 2018.

[Lan11] LANGER, PHILIP: *Adaptable model versioning based on model transformation by demonstration*. Institut für Softwaretechnik und Interaktive Systeme, 2011.

[LBK14] LEE, JAY, BEHRAD BAGHERI und HUNG-AN KAO: *Recent advances and trends of cyber-physical systems and big data analytics in industrial informatics*. In: *International proceeding of int conference on industrial informatics (INDIN)*, Seiten 1–6, 2014.

[LBK15] LEE, JAY, BEHRAD BAGHERI und HUNG-AN KAO: *A cyber-physical systems architecture for industry 4.0-based manufacturing systems*. Manufacturing Letters, 3:18–23, 2015.

[LBL+14] LOCHAU, MALTE, JOHANNES BÜRDEK, SASCHA LITY, MATTHIAS HAGNER, CHRISTOPH LEGAT, URSULA GOLTZ und ANDY SCHÜRR: *Applying Model-based Software Product Line Testing Approaches to the Automation Engineering Domain*. at - Automatisierungstechnik, 62(11):771–780, 2014.

[LCMR10] LE DUC, BAO, PHILIPPE COLLET, JACQUES MALENFANT und NICOLAS RIVIERRE: *A QoI-aware framework for adaptive monitoring*. In: *2nd International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2010)*, Seiten 133–141, 2010.

[Lee06] LEE, EDWARD A.: *Cyber-physical systems-are computing foundations adequate*. In:

Position Paper for NSF Workshop On Cyber-Physical Systems: Research Motivation, Techniques and Roadmap, Band 2, Seiten 1–9, 2006.

[Lee08] LEE, EDWARD A.: *Cyber physical systems: Design challenges*. In: *Object oriented real-time distributed computing (isorc), 2008 11th ieee international symposium on*, Seiten 363–369, 2008.

[Leh79] LEHMAN, MEIR M.: *On understanding laws, evolution, and conservation in the large-program life cycle*. Journal of Systems and Software, 1:213–221, 1979.

[Leh96] LEHMAN, MANNY M.: *Laws of software evolution revisited*. In: *European Workshop on Software Process Technology*, Seiten 108–124, 1996.

[LFA+15] LADIGES, JAN, ALEXANDER FÜLBER, ESTEBAN ARROYO, ALEXANDER FAY, CHRISTOPHER HAUBECK und WINFRIED LAMERSDORF: *Learning material flow models for manufacturing plants from data traces*. In: *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*, Seiten 294–301, 2015.

[LFH+15] LADIGES, JAN, ALEXANDER FAY, CHRISTOPHER HAUBECK, WINFRIED LAMERSDORF, SASCHA LITY und INA SCHAEFER: *Supporting commissioning of production plants by model-based testing and model learning*. In: *Industrial Electronics (ISIE), 2015 IEEE 24th International Symposium on*, Seiten 606–611, 2015.

[LFHL13] LADIGES, JAN, ALEXANDER FAY, CHRISTOPHER HAUBECK und WINFRIED LAMERSDORF: *Operationalized definitions of non-functional requirements on automated production facilities to measure evolution effects with an automation system*. In: *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*, Seiten 1–6, 2013.

[LFL16] LADIGES, JAN, ALEXANDER FAY und WINFRIED LAMERSDORF: *Automated Determining of Manufacturing Properties and Their Evolutionary Changes from Event Traces*. Intelligent Industrial Systems, 2(2):163–178, 2016.

[LHFL14a] LADIGES, JAN, CHRISTOPHER HAUBECK, ALEXANDER FAY und WINFRIED LAMERSDORF: *Evolution management of production facilities by semi-automated requirement verification*. at-Automatisierungstechnik, 62(11):781–793, 2014.

[LHFL14b] LADIGES, JAN, CHRISTOPHER HAUBECK, ALEXANDER FAY und WINFRIED LAMERSDORF: *Semiautomated decision making support for undocumented evolutionary changes*. In: *Workshop Software-Reengineering & Evolution*, 2014.

[LHFL15] LADIGES, JAN, CHRISTOPHER HAUBECK, ALEXANDER FAY und WINFRIED LAMERSDORF: *Learning behaviour models of discrete event production systems from observing input/output signals*. IFAC-PapersOnLine, 48(3):1565–1572, 2015.

[LK07] LAW, AVERILL M. und W. DAVID KELTON: *Simulation modeling and analysis*, Band 3. McGraw-Hill New York, 2007.

[LKKS15] LEE, SEONAH, SUNGWON KANG, SUNGHUN KIM und MATT STAATS: *The impact of view histories on edit recommendations*. IEEE Transactions on Software Engineering, 41(3):314–330, 2015.

[LKY14] LEE, JAY, HUNG-AN KAO und SHANHU YANG: *Service innovation and smart analytics for industry 4.0 and big data environment*. Procedia Cirp, 16:3–8, 2014.

[LL11] LEFEBVRE, DIMITRI und EDOUARD LECLERCQ: *Stochastic Petri net identification for the fault detection and isolation of discrete event systems*. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 41(2):213–225, 2011.

[LLBK13] LEE, JAY, EDZEL LAPIRA, BEHRAD BAGHERI und HUNG-AN KAO: *Recent ad-*

vances and trends in predictive manufacturing systems in big data environment. Manufacturing Letters, 1(1):38–41, 2013.

[LLVH13] LEGAT, CHRISTOPH, STEFFEN LAMPARTER und BIRGIT VOGEL-HEUSER: *Knowledge-based technologies for future factory engineering and control*. In: *Service Orientation in Holonic and Multi Agent Manufacturing and Robotics*, Seiten 355–374. Springer, 2013.

[LMC+14] LEGAT, CHRISTOPH, JAKOB MUND, ALARICO CAMPETELLI, GEORG HACKENBERG, JENS FOLMER, DANIEL SCHÜTZ, MANFRED BROY und BIRGIT VOGEL-HEUSER: *Interface behavior modeling for automatic verification of industrial automation systems' functional conformance*. at-Automatisierungstechnik, 62(11):815–825, 2014.

[LMP06] LORENZOLI, DAVIDE, LEONARDO MARIANI und MAURO PEZZÈ: *Inferring State-based Behavior Models*. In: *Proceedings of the 2006 International Workshop on Dynamic Systems Analysis*, WODA '06, Seiten 25–32, New York, NY, USA, 2006. ACM.

[LNR11] LEGAT, CHRISTOPH, JÖRG NEIDIG und MIKHAIL ROSHCHIN: *Model-based knowledge extraction for automated monitoring and control*. IFAC Proceedings Volumes, 44(1):5225–5230, 2011.

[LP10] LIN, KWEI-JAY und MARK PANAHI: *A real-time service-oriented framework to support sustainable cyber-physical systems*. In: *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, Seiten 15–21, 2010.

[LR+12] LEHNERT, STEFFEN, MATTHIAS RIEBISCH und OTHERS: *A taxonomy of change types and its application in software evolution*. In: *Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on*, Seiten 98–107, 2012.

[LR+13] LEHNERT, STEFFEN, MATTHIAS RIEBISCH und OTHERS: *Rule-based impact analysis for heterogeneous software artifacts*. In: *Software maintenance and reengineering (csmr), 2013 17th european conference on*, Seiten 209–218, 2013.

[LSMR16] LANGHAMMER, MICHAEL, ARMAN SHAHBAZIAN, NENAD MEDVIDOVIC und RALF H. REUSSNER: *Automated extraction of rich software models from limited system information*. In: *Software Architecture (WICSA), 2016 13th Working IEEE/IFIP Conference on*, Seiten 99–108, 2016.

[LST78] LIENTZ, BENNET P., E. BURTON SWANSON und GAIL E. TOMPKINS: *Characteristics of application software maintenance*. Communications of the ACM, 21(6):466–471, 1978.

[Lv92] LIPPE, ERNST und NORBERT VAN OOSTEROM: *Operation-based merging*. In: *ACM SIGSOFT Software Engineering Notes*, Band 17, Seiten 78–87, 1992.

[LWA+13] LADIGES, JAN, IRENEUS WIOR, ESTEBAN ARROYO, ALEXANDER FAY, CHRISTOPHER HAUBECK und WINFRIED LAMERSDORF: *Evolution of production facilities and its impact on non-functional requirements*. In: *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*, Seiten 224–229, 2013.

[LWB+13] LANGER, PHILIP, MANUEL WIMMER, PETRA BROSCH, MARKUS HERRMANNSDÖRFER, MARTINA SEIDL, KONRAD WIELAND und GERTI KAPPEL: *A posteriori operation detection in evolving software models*. Journal of Systems and Software, 86(2):551–566, 2013.

[LXB+16] LI, SIHAN, XUSHENG XIAO, BLAKE BASSETT, TAO XIE und NIKOLAI TILLMANN:

Measuring code behavioral similarity for programming and software engineering education. In: *Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on*, Seiten 501–510, 2016.

[Mah07] MAHMOUD, QUSAY: *Cognitive networks: Towards self-aware networks*. John Wiley & Sons, 2007.

[May01] MAYR, ERNST: *What evolution is*. Science Masters Series, 2001.

[MB05] MOWBRAY, MIRANDA und ALEXANDRE BRONSTEIN: *What kind of self-aware systems does the grid need*. HP Laboratories, 2005.

[MBJ+09] MORIN, BRICE, OLIVIER BARAIS, JEAN-MARC JÉZÉQUEL, FRANCK FLEUREY und ARNOR SOLBERG: *Models@ run. time to support dynamic adaptation*. Computer, 42(10):44–51, 2009.

[MCLM05] MEDA-CAMPAÑA, M. E. und ERNESTO LÓPEZ-MELLADO: *Identification of concurrent discrete event systems using Petri nets*. In: *Proceedings of the 17th IMACS World Congress on Computational and Applied Mathematics*, Seiten 11–15, 2005.

[MD08] MENS, TOM und SERGE DEMEYER: *Software Evolution*. Springer Publishing Company, Incorporated, 1 Auflage, 2008.

[Mer87] MERKLE, RALPH C.: *A digital signature based on a conventional encryption function*. In: *Conference on the theory and application of cryptographic techniques*, Seiten 369–378, 1987.

[MFF14] MAZEL, J., R. FONTUGNE und K. FUKUDA: *A taxonomy of anomalies in backbone network traffic*. IWCMC 2014 - 10th Int. Wireless Communications and Mobile Computing Conf., Seiten 30–36, 2014.

[MFP06] MÜHL, GERO, LUDGER FIEGE und PETER PIETZUCH: *Distributed event-based systems*. Springer Science & Business Media, 2006.

[MGRP09] MORA, BEATRIZ, FELIX GARCIA, FRANCISCO RUIZ und MARIO PIATTINI: *Model-driven software measurement framework: A case study*. In: *Quality Software, 2009. QSIC'09. 9th International Conference on*, Seiten 239–248, 2009.

[Mih14] MIHR, REINER: *An der Kasse – der bleibende Eindruck*, 2014.

[MJB+17] MUND, JAKOB, MAXIMILIAN JUNKER, SAFA BOUGOUFFA, SUHYUN CHA und BIRGIT VOGEL-HEUSER: *Model-based availability analysis for automated production systems: a case study*. In: *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, Seiten 46–55, 2017.

[MJG11] MAGA, CAMELIAR, NASSER JAZDI und PETER GÖHNER: *Reusable models in industrial automation: experiences in defining appropriate levels of granularity*. IFAC Proceedings Volumes, 44(1):9145–9150, 2011.

[MKCC03] MACCORMACK, ALAN, CHRIS F. KEMERER, MICHAEL CUSUMANO und BILL CRANDALL: *Trade-offs between productivity and quality in selecting software development practices*. Ieee Software, 20(5):78–85, 2003.

[MNE15] MAIER, ALEXANDER, OLIVER NIGGEMANN und JENS EICKMEYER: *On the Learning of Timing Behavior for Anomaly Detection in Cyber-Physical Production Systems*. In: *DX@ Safeprocess*, Seiten 217–224, 2015.

[MNNN07] MALEC, JACEK, ANDERS NILSSON, KLAS NILSSON und SLAWOMIR NOWACZYK: *Knowledge-based reconfiguration of automation systems*. In: *2007 IEEE International Conference on Automation Science and Engineering*, Seiten 170–175, 2007.

[Mon14] MONOSTORI, LÁSZLÓ: *Cyber-physical production systems: Roots, expectations and R&D challenges*. Procedia Cirp, 17:9–13, 2014.

[MPP11] MARIANI, L., F. PASTORE und M. PEZZE: *Dynamic Analysis for Diagnosing Integration Faults*. Software Engineering, IEEE Transactions on, 37(4):486–508, 2011.

[MR04] MIRKOVIC, J. und P. REIHER: *A taxonomy of DDoS attack and DDoS defense mechanisms*. SIGCOMM Comp. Comm. Rev., 34(2):39–53, 2004.

[MR09] MARINO, JIM und MICHAEL ROWLEY: *Understanding sca (service component architecture)*. pearson education, 2009.

[MS15] MAINELLI, MICHAEL und MIKE SMITH: *Sharing ledgers for sharing economies: an exploration of mutual distributed ledgers (aka blockchain technology)*. Journal of Financial Perspectives, 3(3), 2015.

[MSB11] MYERS, GLENFORD J., COREY SANDLER und TOM BADGETT: *The art of software testing*. John Wiley & Sons, 2011.

[MSC+18] MO, RAN, WILL SNIPES, YUANFANG CAI, SRINI RAMASWAMY, RICK KAZMAN und MARTIN NAEDELE: *Experiences applying automated architecture analysis tool suites*. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, Seiten 779–789, 2018.

[MVLZ11] MERDAN, MUNIR, MATHIEU VALLEE, WILFRIED LEPUSCHITZ und ALOIS ZOITL: *Monitoring and diagnostics of industrial systems using automation agents*. International journal of production research, 49(5):1497–1509, 2011.

[MvZB08] MOONEN, LEON, ARIE VAN DEURSEN, ANDY ZAIDMAN und MAGIEL BRUNTINK: *On the interplay between software testing and evolution and its effect on program comprehension*. In: *Software evolution*, Seiten 173–202. Springer, 2008.

[NFM17] NEGRI, ELISA, LUCA FUMAGALLI und MARCO MACCHI: *A review of the roles of digital twin in cps-based production systems*. Procedia Manufacturing, 11:939–948, 2017.

[NL15] NIGGEMANN, OLIVER und VOLKER LOHWEG: *On the diagnosis of cyber-physical production systems: state-of-the-art and research agenda*. Austin, Texas, USA, Association for the Advancement of Artificial Intelligence, 2015.

[NLM18] NAM, DAYE, YOUN KYU LEE und NENAD MEDVIDOVIC: *EVA: a tool for visualizing software architectural evolution*. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, Seiten 53–56, 2018.

[NSMŠ15] NOVÁK, PETR, ESTEFAN\'\IA SERRAL, RICHARD MORDINYI und RADEK ŠINDELÁŘ: *Integrating heterogeneous engineering knowledge and tools for efficient industrial simulation model support*. Advanced Engineering Informatics, 29(3):575–590, 2015.

[NSW+02] NIERE, JÖRG, WILHELM SCHÄFER, JÖRG P. WADSACK, LOTHAR WENDEHALS und JIM WELSH: *Towards pattern-based design recovery*. In: *Proceedings of the 24th international conference on Software engineering*, Seiten 338–348, 2002.

[NW70] NEEDLEMAN, SAUL B. und CHRISTIAN D. WUNSCH: *A general method applicable to the search for similarities in the amino acid sequence of two proteins*. Journal of molecular biology, 48(3):443–453, 1970.

[Obj12] OBJECT MANAGEMENT GROUP: *Architecture-Driven Modernization: Structured Metrics Meta-Model*, 2012.

[OLA+] OTTO, BORIS, STEFFEN LOHMANN, SÖREN AUER, JAN CIRULLIES, ANDREAS EITEL, THILO ERNST, CHRISTIAN HAAS, MANUEL HUBER, JAN JÜRJENS, CHRISTOPH LANGE, CHRISTIAN MADER, NADJA MENZ, RALF NAGEL, HEINRICH PETTENPOHL, JAROSLAV PULLMANN, CHRISTOPH QUIX, JOCHEN SCHON, DANIEL SCHULZ, JULIAN SCHÜTTE und SVEN WENZEL: *Reference Architecture Model for the Industrial Data Space*.

[OM05] O'BRIEN, JAMES A. und GEORGE M. MARAKAS: *Introduction to information systems*, Band 13. McGraw-Hill/Irwin New York City, USA, 2005.

[Ors17] ORSINI, GABRIEL: *Kontextadaptive Anwendungsarchitekturen für das mobile Cloud Computing*. Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, 2017.

[OSL11] ONORI, MAURO, DANIEL SEMERE und BENGT LINDBERG: *Evolvable systems: an approach to self-X production*. International Journal of Computer Integrated Manufacturing, 24(5):506–516, 2011.

[OvKV11] OKANOVIĆ, DUV SAN, ANDRÉ VAN HOORN, ZORA KONJOVIĆ und MILAN VIDAKOVIĆ: *Towards adaptive monitoring of Java EE applications*. In: *The 5th International Conference on Information Technology*, 2011.

[Par94] PARNAS, DAVID LORGE: *Software aging*. In: *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*, Seiten 279–287, 1994.

[PB09] POKAHR, ALEXANDER und LARS BRAUBACH: *From a Research to an Industrial-Strength Agent Platform: Jadex V2*. In: HANS ROBERT HANSEN, DIMITRIS KARAGIANNIS, HANS-GEORG FILL (Herausgeber): *Business Services: Konzepte, Technologien, Anwendungen - 9. Internationale Tagung Wirtschaftsinformatik (WI 2009)*, Seiten 769–778. Österreichische Computer Gesellschaft Wien, 2009.

[PB11] POKAHR, ALEXANDER und LARS BRAUBACH: *Active components: A software paradigm for distributed systems*. In: *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02*, Seiten 141–144, 2011.

[PB13] POKAHR, ALEXANDER und LARS BRAUBACH: *The active components approach for distributed systems development*. International Journal of Parallel, Emergent and Distributed Systems, 28(4):321–369, 2013.

[PBHL14] POKAHR, ALEXANDER, LARS BRAUBACH, CHRISTOPHER HAUBECK und JAN LADIGES: *Programming BDI agents with pure Java*. In: *German Conference on Multiagent System Technologies*, Seiten 216–233, 2014.

[PG02] PINZGER, MARTIN und HARALD GALL: *Pattern-supported architecture recovery*. In: *Program Comprehension, 2002. Proceedings. 10th International Workshop on*, Seiten 53–61, 2002.

[PGF08] PINZGER, MARTIN, HARALD GALL und MICHAEL FISCHER: *Software Evolution Analysis and Visualization*. Emerging Methods, Technologies, and Process Management in Software Engineering, Seiten 177–200, 2008.

[PKBL18] POSDORFER, WOLF, JULIAN KALINOWSKI, HEIKO BORNHOLDT und WINFRIED LAMERSDORF: *Decentralized Billing and Subcontracting of Application Services for Cloud Environment Providers*. In: *WESOACS2018*, Lecture Notes in Computer Science. Springer Verlag, 2018.

[PKH+18] PIETSCH, CHRISTOPHER, UDO KELTER, CHRISTOPHER HAUBECK, WINFRIED LAMERSDORF, ABHISHEK CHAKRABORTY und ALEXANDER FAY: *Using Model*

Differencing to reason about Observable Behaviour Changes of Manufacturing Systems. at-Automatisierungstechnik, 2018.

[PKK+15] PIETSCH, CHRISTOPHER, TIMO KEHRER, UDO KELTER, DENNIS REULING und MANUEL OHRNDORF: *SiPL–A Delta-Based Modeling Framework for Software Product Line Engineering*. In: *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, Seiten 852–857, 2015.

[POvG18] PITAKRAT, TEERAT, DUŠAN OKANOVIĆ, ANDRÉ VAN HOORN und LARS GRUNSKE: *Hora: Architecture-aware online failure prediction*. Journal of Systems and Software, 137:669–685, 2018.

[PPS+05] PIRTTIOJA, TEPPO, ANTTI PAKONEN, ILKKA SEILONEN, AARNE HALME und KARI KOSKINEN: *Multi-agent based information access services for condition monitoring in process automation*. In: *Proc. INDIN'05, 3rd IEEE International Conference on Industrial Informatics*, Seiten 240–245, 2005.

[PSG14] PRÄHOFER, HERBERT, ROLAND SCHATZ und ANDREAS GRIMMER: *Behavioral model synthesis of PLC programs from execution traces*. In: *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, Seiten 1–5, 2014.

[RAVH16] REGULIN, DANIEL, THOMAS AICHER und BIRGIT VOGEL-HEUSER: *Improving transferability between different engineering stages in the development of automated material flow modules*. Transactions on Automation Science and Engineering, 13(4):1422–1432, 2016.

[RB02] RAUSCH, ANDREAS und MANFRED BROY: *Evolutionary development of software architectures*. Technology for Evolutionary Software Development, Seiten 16–33, 2002.

[RB14] REICHELT, DAVID GEORG und LARS BRAUBACH: *Sicherstellung von Performanzeigenschaften durch kontinuierliche Performanztests mit dem KoPeMe Framework*. In: *Software Engineering*, Seiten 119–124, 2014.

[RBB+] REUSSNER, RALF, STEFFEN BECKER, ERIK BURGER, JENS HAPPE, MICHAEL HAUCK, ANNE KOZIOLEK, HEIKO KOZIOLEK, KLAUS KROGMANN und MICHAEL KUPERBERG: *The Palladio component model*.

[RBJ17] RUMBAUGH, JAMES, GRADY BOOCH und IVAR JACOBSON: *The unified modeling language reference manual*. Addison Wesley, 2017.

[Rei17] REICHERT, KIM: *Erkennung von Anomalien in Verteilten Systemen: Eine Frage von Metriken*. Masterarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, 2017.

[RLL10] ROTH, MATTHIAS, JEAN-JACQUES LESAGE und LOTHAR LITZ: *Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems*. In: *American Control Conference (ACC), 2010*, Seiten 2601–2606, 2010.

[RLL11] ROTH, MATTHIAS, JEAN-JACQUES LESAGE und LOTHAR LITZ: *The concept of residuals for fault localization in discrete event systems*. Control Engineering Practice, 19(9):978–988, 2011.

[Rob06] ROBINSON, WILLIAM N.: *A requirements monitoring framework for enterprise systems*. Requirements engineering, 11(1):17–41, 2006.

[Roh15] ROHR, MATTHIAS: *Workload-sensitive timing behavior analysis for fault localization in software systems*. BoD–Books on Demand, 2015.

[RPH+17] REICHERT, KIM, ALEXANDER POKAHR, TILL HOHENBERGER, CHRISTOPHER HAUBECK und WINFRIED LAMERSDORF: *A Taxonomy of Anomalies in Distributed*

Cloud Systems: The CRI-Model. In: *International Symposium on Intelligent and Distributed Computing*, Seiten 247–261, 2017.

[RRW14] RINGERT, JAN OLIVER, BERNHARD RUMPE und ANDREAS WORTMANN: *From software architecture structure and behavior modeling to implementations of cyber-physical systems*. arXiv preprint arXiv:1408.5690 [Titel anhand dieser ArXiv-ID in Citavi-Projekt übernehmen], 2014.

[RSLR15] ROSENDAHL, RONALD, NICOLE SCHMIDT, ARNDT LÜDER und DARIA RYASHENTSEVA: *Industry 4.0 value networks in legacy systems*. In: *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, Seiten 1–4, 2015.

[RT01] RYDER, BARBARA G. und FRANK TIP: *Change impact analysis for object-oriented programs*. In: *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, Seiten 46–53, 2001.

[RTF15] RACCHETTI, LORENZO, LORENZO TACCONI und CESARE FANTUZZI: *Generating automatically the documentation from PLC code by D4T3 to improve the usability and life cycle management of software in automation*. In: *Automation Science and Engineering (CASE), 2015 IEEE International Conference on*, Seiten 168–173, 2015.

[RUPVH15] RÖSCH, SUSANNE, SEBASTIAN ULEWICZ, JULIEN PROVOST und BIRGIT VOGEL-HEUSER: *Review of model-based testing approaches in production automation and adjacent domains-current challenges and research gaps*. Journal of Software Engineering and Applications, 2015.

[RvM+08] ROHR, MATTHIAS, ANDRÉ VAN HOORN, JASMINKA MATEVSKA, NILS SOMMER, LENA STOEVER, SIMON GIESECKE und WILHELM HASSELBRING: *Kieker: Continuous monitoring and on demand visualization of Java software behavior*. In: *Proceedings of the IASTED International Conference on Software Engineering 2008*, Seiten 80–85. ACTA Press, 2008.

[RWZ10] ROBILLARD, MARTIN, ROBERT WALKER und THOMAS ZIMMERMANN: *Recommendation systems for software engineering*. Ieee Software, 27(4):80–86, 2010.

[SBB+10] SCHAEFER, INA, LORENZO BETTINI, VIVIANA BONO, FERRUCCIO DAMIANI und NICO TANZARELLA: *Delta-oriented programming of software product lines*. In: *International Conference on Software Product Lines*, Seiten 77–91, 2010.

[SBW99] SZYPERSKI, CLEMENS, JAN BOSCH und WOLFGANG WECK: *Component-oriented programming*. In: *European Conference on Object-Oriented Programming*, Seiten 184–192, 1999.

[SDHM03] STÅLHANE, TOR, TORGEIR DINGSØYR, GEIR KJETIL HANSSEN und NILS BREDE MOE: *Post mortem–an assessment of two approaches*. In: *Empirical Methods and Studies in Software Engineering*, Seiten 129–141. Springer, 2003.

[SDK+11] STAMMEL, JOHANNES, ZOYA DURDIK, KLAUS KROGMANN, ROLAND WEISS und HEIKO KOZIOLEK: *Software evolution for industrial automation systems: Literature overview*. KIT, Fakultät für Informatik, 2011.

[SG08] SCHOR, WOLFGANG und NORBERT GROSSE: *Hierarchien und Symbolik in der Produktionstechnik*. atp edition, 50(11):66–74, 2008.

[SH16] SCHUHMACHER, JAN und VERA HUMMEL: *Decentralized control of logistic processes in cyber-physical production systems at the example of ESB Logistics Learning Factory*. Procedia Cirp, 54:19–24, 2016.

[SHC+10] SONG, HUI, GANG HUANG, FRANCK CHAUVEL, YANCHUN SUN und HONG MEI: *SM@ RT: representing run-time system data as MOF-compliant models*. In: *2010 ACM/IEEE 32nd International Conference on Software Engineering*, Band 2, Seiten 303–304, 2010.

[SHEA10] SANTAMBROGIO, MARCO D., HENRY HOFFMANN, JONATHAN EASTEP und ANANT AGARWAL: *Enabling technologies for self-aware adaptive systems*. In: *Adaptive Hardware and Systems (AHS), 2010 NASA/ESA Conference on*, Seiten 149–156, 2010.

[SKCD16] SOSKA, KYLE, ALBERT KWON, NICOLAS CHRISTIN und SRINIVAS DEVADAS: *Beaver: A Decentralized Anonymous Marketplace with Secure Reputation*. IACR Cryptology ePrint Archive, 2016:464, 2016.

[SKK+12] SZTIPANOVITS, JANOS, XENOFON KOUTSOUKOS, GABOR KARSAI, NICHOLAS KOTTENSTETTE, PANOS ANTSAKLIS, VIJAY GUPTA, BILL GOODWINE, JOHN BARAS und SHIGE WANG: *Toward a science of cyber–physical system integration*. Proceedings of the IEEE, 100(1):29–44, 2012.

[SL10] SCHACKMANN, HOLGER und HORST LICHTER: *Metrik-basierte Auswertung von Software-Entwicklungsarchiven zur Prozessbewertung*, Band 7. Citeseer, 2010.

[SLS+15] SCHLEIPEN, MIRIAM, ARNDT LÜDER, OLAF SAUER, HOLGER FLATT und JÜRGEN JASPERNEITE: *Requirements and concept for plug-and-work*. at-Automatisierungstechnik, 63(10):801–820, 2015.

[SLV03] STOERMER, CHRISTOPH, O. LIAM und CHRIS VERHOEF: *Moving towards quality attribute driven software architecture reconstruction*. In: *null*, Seite 46, 2003.

[Som12] SOMMERVILLE, IAN: *Software engineering*. Addison-wesley, 9. Auflage Auflage, 2012.

[SPH+06] SEILONEN, ILKKA, TEPPO PIRTTIOJA, AARNE HALME, KARI KOSKINEN und ANTTI PAKONEN: *Indirect process monitoring with constraint handling agents*. In: *2006 4th IEEE International Conference on Industrial Informatics*, Seiten 1323–1328, 2006.

[SPT19] SCHNEIDER, GEORG FERDINAND, GEORG AMBROSIUS PESSLER und WALTER TERKAJ: *Knowledge-based Conversion of Finite State Machines in Manufacturing Automation*. Procedia Manufacturing, 28:189–194, 2019.

[SRB17] SEIDENSTRICKER, SVEN, ERWIN RAUCH und CINZIA BATTISTELLA: *Business model engineering for distributed manufacturing systems*. Procedia Cirp, 62:135–140, 2017.

[SS90] SETHI, ANDREA KRASA und SURESH PAL SETHI: *Flexibility in manufacturing: a survey*. International journal of flexible manufacturing systems, 2(4):289–328, 1990.

[SS13] SNEED, HARRY und RICHARD SEIDL: *Softwareevolution: Erhaltung und Fortschreibung bestehender Softwaresysteme*. dPunkt Verlag, Heidelberg, 2013.

[SSA14] SEIDL, CHRISTOPHE, CHRISTOPHE SCHAEFER und UWE ASSMANN: *DeltaEcore - A Model-Based Delta Language Generation Framework*. In: *Modellierung*, 2014.

[ST09] SALEHIE, MAZEIAR und LADAN TAHVILDARI: *Self-adaptive software: Landscape and research challenges*. ACM transactions on autonomous and adaptive systems (TAAS), 4(2):14, 2009.

[Sta17] STAMMEL, JOHANNES JOSEF: *Architekturbasierte Bewertung und Planung von Änderungsanfragen*, Band 19. KIT Scientific Publishing, 2017.

[Sub18] SUBRAMANIAN, HEMANG: *Decentralized blockchain-based electronic marketplaces*. Commun. ACM, 61(1):78–84, 2018.

[SW02] SMITH, CONNIE U. und LLOYD G. WILLIAMS: *Performance solutions: a practical guide to creating responsive, scalable software*, Band 1. Addison-Wesley Reading, 2002.

[SWYS11] SHI, JIANHUA, JIAFU WAN, HEHUA YAN und HUI SUO: *A survey of cyber-physical systems*. In: *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, Seiten 1–6, 2011.

[Tan12] TANCHOCO, J. M.: *Material flow systems in manufacturing*. Springer Science & Business Media, 2012.

[TBC+14] TÖRNGREN, M., S. BENSALEM, M. V. CENGARLE, J. MCDERMID, R. PASSERONE und A. SANGIOVANNI-VINCENTELLI: *Cyber-physical european roadmap & Strategy*. Cyber-Physical European Roadmap and Strategy D5. 1, Tech. Rep., 2014.

[TC13] TOBERGTE, DAVID R. und SHIRLEY CURTIS: *Why Internet services fail and what can be done about these.* Journal of Chemical Information and Modeling, 53(9):1689–1699, 2013.

[TEL11] TORENS, CHRISTOPH, LARS EBRECHT und KARSTEN LEMMER: *Inverse Model Based Testing–Generating Behavior Models from Abstract Test Cases*. In: *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, Seiten 75–78, 2011.

[TH99] TRAN, JOHN B. und RICHARD C. HOLT: *Forward and reverse repair of software architecture*. In: *Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research*, Seite 12, 1999.

[TK18] TAMBURRI, DAMIAN A. und RICK KAZMAN: *General methods for software architecture recovery: a potential approach and its evaluation*. Empirical Software Engineering, 23(3):1457–1489, 2018.

[TN14] TRIPATHY, PRIYADARSHI und KSHIRASAGAR NAIK: *Software Evolution and Maintenance: A Practitioner's Approach*. John Wiley & Sons, 2014.

[TRdL07] TORCHIANO, MARCO, FILIPPO RICCA und ANDREA DE LUCIA: *Empirical studies in software maintenance and evolution*. In: *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, Seiten 491–494, 2007.

[TVG09] TAN, YING, MEHMET C. VURAN und STEVE GODDARD: *Spatio-temporal event model for cyber-physical systems*. In: *Distributed Computing Systems Workshops, 2009. ICDCS Workshops' 09. 29th IEEE International Conference on*, Seiten 44–50, 2009.

[TW15] TURAU, VOLKER und CHRISTOPH WEYER: *Algorithmische Graphentheorie*. Walter de Gruyter GmbH & Co KG, 2015.

[UL06] UTTING, MARK und BRUNO LEGEARD: *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc, 2006.

[van11] VAN DER AALST, WIL M. P.: *Process Discovery: An Introduction*. In: *Process Mining*, Seiten 125–156. Springer, 2011.

[Ver14] VEREIN DEUTSCHER INGENIEURE: *VDI2206:2014, Entwicklungsmethodik für mechatronische Systeme*, 2014.

[VH13] VALILAI, OMID FATAHI und MAHMOUD HOUSHMAND: *A collaborative and in-*

tegrated platform to support distributed manufacturing system using a service-oriented approach based on cloud computing paradigm. Robotics and computer-integrated manufacturing, 29(1):110–127, 2013.

[VHDF+14] VOGEL-HEUSER, BIRGIT, CHRISTIAN DIEDRICH, ALEXANDER FAY, SABINE JESCHKE, STEFAN KOWALEWSKI, MARTIN WOLLSCHLAEGER und PETER GÖHNER: *Challenges for software engineering in automation*. Journal of Software Engineering and Applications, 7(05):440, 2014.

[VHFF+15] VOGEL-HEUSER, BIRGIT, STEFAN FELDMANN, JENS FOLMER, MATTHIAS KOWAL, INA SCHAEFER, JAN LADIGES, ALEXANDER FAY, CHRISTOPHER HAUBECK, WINFRIED LAMERSDORF, SASCHA LITY, TIMO KEHRER, MATTHIAS TICHY, SINEM GETIR, ULBRICH MATTHIAS, VLADIMIR KLEBANOV und BERNHARD BECKERT: *Selected challenges of software evolution for automated production systems*. In: *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*, Seiten 314–321, 2015.

[VHFST15] VOGEL-HEUSER, BIRGIT, ALEXANDER FAY, INA SCHAEFER und MATTHIAS TICHY: *Evolution of software in automated production systems: Challenges and research directions*. Journal of Systems and Software, 110:54–84, 2015.

[VHHC+17] VOGEL-HEUSER, BIRGIT, ROBERT HEINRICH, SUHYUN CHA, KIANA ROSTAMI, FELIX OCKER, SANDRO KOCH, RALF REUSSNER und SIMON ZIEGLTRUM: *Maintenance effort estimation with KAMP4aPS for cross-disciplinary automated PLC-based Production Systems-a collaborative approach*. IFAC-PapersOnLine, 50(1):4360–4367, 2017.

[VHLFF] VOGEL-HEUSER, BIRGIT, CHRISTOPH LEGAT, JENS FOLMER und STEFAN FELDMANN: *Researching evolution in industrial plant automation: Scenarios and documentation of the pick and place unit*.

[vRH+09] VAN HOORN, ANDRÉ, MATTHIAS ROHR, WILHELM HASSELBRING, JAN WALLER, JENS EHLERS, SÖREN FREY und DENNIS KIESELHORST: *Continuous monitoring of software services: Design and application of the Kieker framework*. Doktorarbeit, Kiel University, 2009.

[VRV+10] VAN DER AALST, WIL MP, VLADIMIR RUBIN, H. M.W. VERBEEK, BOUDEWIJN F. VAN DONGEN, EKKART KINDLER und CHRISTIAN W. GÜNTHER: *Process mining: a two-step approach to balance between underfitting and overfitting*. Software & Systems Modeling, 9(1):87, 2010.

[Vya13] VYATKIN, VALERIY: *Software engineering in industrial automation: State-of-the-art review*. Transactions on Industrial Informatics, 9(3):1234–1249, 2013.

[Wal15] WALLER, JAN: *Performance Benchmarking of Application Monitoring Frameworks*. BoD–Books on Demand, 2015.

[Wal18] WALTER, JÜRGEN: *Automation in Software Performance Engineering Based on a Declarative Specification of Concern*. Doktorarbeit, University of Würzburg, Germany, 2018.

[WBÖ09] WINKLER, DIETMAR, STEFAN BIFFL und THOMAS ÖSTREICHER: *Test-Driven Automation: Adopting Test-First Development to Improve Automation Systems Engineering Processes*. In: *EuroSPI'09*, 2009.

[WC10] WILLIAMS, BYRON J. und JEFFREY C. CARVER: *Characterizing software architecture changes: A systematic review*. Information and Software Technology, 52(1):31–51, 2010.

[WG11] WERNER, EDITH und JENS GRABOWSKI: *Model reconstruction: mining test cases*. In: *Proc. of 3rd Int. Conf. on Advances in System Testing and Validation Lifecycle*, Seiten 97–102, 2011.

[WGG13] WÜRSCH, MICHAEL, EMANUEL GIGER und HARALD C. GALL: *Evaluating a query framework for software evolution data*. Transactions on Software Engineering and Methodology, 22(4):38, 2013.

[Wil13] WILFREDO BOHORQUEZ LOPEZ, VICTOR AND ESTEVES, JOSE: *Acquiring external knowledge to avoid wheel re-invention*. Journal of Knowledge Management, 17(1):87–105, 2013.

[Win95] WINKLER, WILLIAM E.: *Matching and record linkage*. Business survey methods, 1:355–384, 1995.

[WJ95] WOOLDRIDGE, MICHAEL und NICHOLAS R. JENNINGS: *Intelligent agents: Theory and practice*. The knowledge engineering review, 10(2):115–152, 1995.

[WKT11] WU, FANG-JING, YU-FEN KAO und YU-CHEE TSENG: *From wireless sensor networks towards cyber physical systems*. Pervasive and Mobile computing, 7(4):397–413, 2011.

[Wol09] WOLF, WAYNE: *Cyber-physical systems*. Computer, 42(3):88–89, 2009.

[WSKK17] WALTER, JÜRGEN, CHRISTIAN STIER, HEIKO KOZIOLEK und SAMUEL KOUNEV: *An Expandable Extraction Framework for Architectural Performance Models*. In: *Proceedings of the 3rd International Workshop on Quality-Aware DevOps (QUDOS'17)*. ACM, 2017.

[WTO15] WANG, LIHUI, MARTIN TÖRNGREN und MAURO ONORI: *Current status and advancement of cyber-physical systems in manufacturing*. Journal of Manufacturing Systems, 37:517–527, 2015.

[WTRS12] WU, DAZHONG, J. LANE THAMES, DAVID W. ROSEN und DIRK SCHAEFER: *Towards a cloud-based design and manufacturing paradigm: looking backward, looking forward*. In: *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Seiten 315–328, 2012.

[WW18] WANG, LIHUI und XI VINCENT WANG: *Cloud-Based Cyber-Physical Systems in Manufacturing*. Springer, 2018.

[XTKM07] XIE, TAO, KUNAL TANEJA, SHREYAS KALE und DARKO MARINOV: *Towards a framework for differential unit testing of object-oriented programs*. In: *Automation of Software Test, 2007. AST'07. Second International Workshop on*, Seite 5, 2007.

[YPW08] YILMAZ, CEMAL, AMIT PARADKAR und CLAY WILLIAMS: *Time will tell: fault localization using time spectra*. In: *Proceedings of the 30th international conference on Software engineering*, Seiten 81–90, 2008.

[ZQLL17] ZHANG, YINGFENG, CHENG QIAN, JINGXIANG LV und YING LIU: *Agent and cyber-physical system based self-organizing and self-adaptive intelligent shopfloor*. IEEE Transactions on Industrial Informatics, 13(2):737–747, 2017.

[ZT16] ZHANG, YINGFENG und FEI TAO: *Optimization of Manufacturing Systems Using the Internet of Things*. Academic Press, 2016.

[ZZCH10] ZHANG, SHUAI, SHUFEN ZHANG, XUEBIN CHEN und XIUZHEN HUO: *Cloud computing research and development trend*. In: *Future Networks, 2010. ICFN'10. Second International Conference on*, Seiten 93–97, 2010.

Darstellungsverzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Verfahrensvorschriftsverzeichnis

Quellcodeverzeichnis

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den 12. März 2019

Christopher Haubeck