Absicherung von verteilten Automatisierungssystemen nach Änderungen der Steuerungssoftware

–

Modellkomposition zur Nutzung der funktionalen Verifikation

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von
Andreas Zeller
aus Leonberg

Hauptberichter:	Prof. Dr.-Ing. Dr. h. c. Michael Weyrich
Mitberichter:	Prof. Dr.-Ing. Christian Diedrich
Tag der Einreichung:	04.04.2019
Tag der mündlichen Prüfung:	

Institut für Automatisierungstechnik und Softwaresysteme
der Universität Stuttgart

2019

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Automatisierungstechnik und Softwaresysteme (IAS) der Universität Stuttgart.

Stuttgart, im April 2019

Andreas Zeller

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ARIS	**Ar**chitektur integrierter **I**nformations**s**ysteme
ATS	**A**utoma**t**isierungs**s**ystem
BPEL	**B**usiness **P**rocess **E**xecution **L**anguage
BPMN	**B**usiness **P**rocess **M**odel and **N**otation
CTL	**C**omputation **T**ree **L**ogic
FMEA	**F**ehler**m**öglichkeits- und -**e**influss**a**nalyse
GRAFCET	**Gra**phe **F**onctionnel de **C**ommande **E**tapes/**T**ransitions
JDBC	**J**ava **D**ata**b**ase **C**onnectivity
JSON	**J**ava**s**cript **O**bject **N**otation
JSP	**J**ava**S**erver **P**ages
M2M	**M**achine-**to**-**M**achine
NCES	**N**et **C**ondition/**E**vent **S**ystems
LAN	**L**ocal **A**rea **N**etwork
LTL	**L**inear **T**emporal **L**ogic
OASIS	**O**rganization for the **A**dvancement of **S**tructured **I**nformation **S**tandards
OMG	**O**bject **M**anagement **G**roup
OPC UA	**O**pen **P**latform **C**ommunications **U**nified **A**rchitecture
PIPN	**P**rozessbezogen **i**nterpretierte **P**etri-**N**etze
PNML	**P**etri **N**et **M**arkup **L**anguage
QS	**Q**ualitäts**s**icherung
R-TNCES	**R**econfigurable **T**imed **N**et **C**ondition/**E**vent **S**ystem
SFC	**S**equential **F**unction **C**hart

SIPN	**S**teuerungstechnisch **i**nterpretierte **P**etri-**N**etze
SPENAT	**S**icheres **Pe**tri-**N**etz mit **At**tributen
SysML	**Sys**tems **M**odeling **L**anguage
SOA	**S**ervice**o**rientierte **A**rchitektur
TestIAS	**Test**system am **I**nstitut für **A**utomatisierungstechnik und **S**oftwaresysteme
TTCN-3	**T**esting and **T**est **C**ontrol **N**otation
TCP/IP	**T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol
UML	**U**nified **M**odeling **L**anguage
UML-Marte	**U**nified **M**odeling **L**anguage – **M**odeling and **A**nalysis of **R**eal **T**ime and **E**mbedded systems
UML-PA	**U**nified **M**odeling **L**anguage – **P**lant **A**utomation
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
XML	E**x**tensible **M**arkup **L**anguage

Begriffsverzeichnis

Abhängigkeitsmodell: Darstellung von Abhängigkeiten zwischen Komponenten eines Systems.

Anlagenbetreiber: Person, die die Gesamtverantwortung für den sicheren Betrieb der Anlage trägt und die Regeln und Randbedingungen der Organisation vorgibt (nach [1]).

Asynchrone Kommunikation: Kommunikationsmodus, bei dem Senden und Empfangen von Daten zeitlich versetzt auftreten können. Dies muss nicht dazu führen, dass der Sender blockiert ist, bis er eine Antwort erhält.

Auswirkungsanalyse: Die Untersuchung und Auswertung der Auswirkungen einer Änderung an einer Komponente auf die Gültigkeit funktionaler Anforderungen anderer Komponenten.

Automatisierungssystem: Ein System mit dem Ziel der Automatisierung technischer Prozesse [2].

Cyber-physisches Produktionssystem: Ist ein Cyber-physisches System, das in der Produktion eingesetzt wird. Cyber-physisch beschreibt dabei, dass reale (physische) Objekte und Prozesse mit informationsverarbeitenden (virtuellen) Objekten und Prozessen über offene, teilweise globale Informationsnetze verbunden sind [3].

Digitaler Zwilling: Digitale Repräsentanz eines Objekts aus der realen Welt. Der Digitale Zwilling enthält Informationen des Objekts aus dessen gesamten Lebenszyklus. Diese Informationen können unter anderem Daten, Modelle und Algorithmen sein.

DOT: Ein textbasiertes Dateiformat zur Beschreibung der visuellen Darstellung von Modellen.

Fehlermöglichkeits- und Einfluss-Analyse: Ein systematischer Ansatz zur Risikoidentifikation sowie zur Analyse möglicher Fehler(aus)wirkungen und zur Vermeidung von Fehlern [4].

Flexible Produktionssysteme: Integriertes System rechnergesteuerter Maschinenmodule und Materialtransporteinrichtungen für die automatisierte, zufallsgesteuerte Verarbeitung von Teilen. Das Ziel ist es, vordefinierte Produktfamilien, die sich ändern können, kosteneffizient mit geforderter Qualität und Quantität zu produzieren [5].

Horizontale Integration: Integration innerhalb einer funktionalen/organisatorischen Hierarchieebene über Systemgrenzen hinweg [3].

Industrie 4.0: Zukunftsprojekt zur Förderung der Digitalisierung in der industriellen Produktion.

Komponente: Eine Software-Komponente ist ein Software-Baustein, der konform zu einem Komponentenmodell ist und nach einem Composition-Standard ohne Änderungen mit anderen Komponenten verknüpft, ausgeführt und wiederverwendet werden kann [6].

Komposition: Zusammenfügen mehrerer Modelle zu einem Gesamtmodell über definierte Schnittstellen.

Lebenszyklus: Folge von Prozessen, die ein Gegenstand oder eine Komponente von der Entstehung bis zum Vergehen durchläuft.

Lose Kopplung: Ziel der losen Kopplung ist die Reduktion der Abhängigkeit zwischen Software-Komponenten. Die Komponenten binden sich zur Laufzeit und besitzen einen azyklischen Kommunikationsstil.

Modell: Vereinfachtes Abbild der Realität unter einer bestimmten Sicht [7].

Orchestrierung von Services: Flexibles Verbinden von einzelnen Diensten für einen definierten Zweck. Anmerkung: Dies kann während der Planungsphase und/oder zur Laufzeit erfolgen [8].

Over-the-air (OTA) Updates: Aufspielen neuer Software-Versionen über drahtlose Kommunikationsnetzwerke.

Platzhalter: Eine rudimentäre Modellierung einer Komponente oder eines Systems.

Redundanz: Verwendung mehrerer Elemente oder Systeme zur Durchführung der gleichen Funktion [9].

Regressionstest: Erneutes Testen eines bereits getesteten Programms bzw. einer Teilfunktionalität nach deren Änderung. Ziel ist es, nachzuweisen, dass durch die vorgenommenen Änderungen keine Fehlerzustände eingebaut oder bisher maskierte Fehlerzustände freigelegt wurden. Anmerkung: Ein Regressionstest wird durchgeführt, wenn die Software oder ihre Umgebung verändert wurde [10].

Rekonfigurierbare Produktionssysteme: Rekonfigurierbare Produktionssysteme sind so konzipiert, dass eine schnelle Strukturänderung möglich ist, um zügige Anpassungen der Produktionskapazität und Funktionalität gemäß geänderter Marktanforderungen durchzuführen [5].

Safety: Abwesenheit von Gefahren die von einem System auf die Umwelt ausgehen.

Schnittstelle: Definierte Verbindungsstelle einer Funktionseinheit, über die diese mit anderen Funktionseinheiten verbunden werden kann [3].

Service: (deutsch: Dienst) Abgegrenzter Funktionsumfang, der von einer Entität oder Organisation über Schnittstellen angeboten wird [11].

Serviceorientierung: Paradigma, welches das einfache Austauschen, Hinzufügen und Entfernen von lose gekoppelten Services ermöglicht [8].

Serviceorientierte Architektur: Paradigma zur Nutzung und Organisation von Services in verteilten informationstechnischen Systemen.

Software-Änderung: Modifikation des Programmcodes einer Software.

Steuerungssystem: Einheit zur Beeinflussung eines technischen Prozesses. Dabei werden ausgehend von Sensordaten und dem inneren Zustand des Steuerungssystems Aktorsignale berechnet.

Strukturmodell: Beschreibung, um die Struktur von Systemen abstrakt darzustellen.

System: Menge von Komponenten, die in Beziehung stehen [3].

Technischer Prozess: Gesamtheit aufeinander einwirkender Vorgänge eines technischen Systems, durch welche Materie, Energie oder Information umgeformt, transportiert oder gespeichert werden und dessen physikalische Größen mit technischen Mitteln erfasst und beeinflusst werden können [7].

Validierung: Überprüfung, ob eine Software den Praxisanforderungen entspricht. Sie stellt fest, ob sie dem jeweiligen Einsatzzweck genügt.

Verhaltensmodell: Beschreibung, um das Verhalten von Systemen abstrakt darzustellen.

Verifikation: Überprüfung, ob eine Software ihrer Spezifikation entspricht. Zur Feststellung, ob nach Erstellung der Anforderungen Fehler gemacht wurden.

Verteiltes System: „Ein verteiltes System ist definiert durch eine Menge von Funktionen oder Komponenten, die in Beziehung zueinander stehen (Client-Server-Beziehung) und eine Funktion erbringen, die nicht erbracht werden kann durch die Komponenten alleine“ [12].

Vertikale Integration: Integration innerhalb eines Systems über funktionale/organisatorische Hierarchie-Ebenen hinweg [3].

Wohldefinierte Schnittstelle: Syntaktisch wie semantisch definierte Schnittstelle.

Zusammenfassung

Wirtschaftliche und technologische Treiber führen dazu, dass sich Steuerungen von Automatisierungssystemen zunehmend zu komplexen Softwaresystemen wandeln. Dabei ermöglicht Software die Realisierung komplexer Steuerungsaufgaben und, aufgrund der leichten Änderbarkeit, ein wandlungsfähiges Steuerungssystem. Jede Änderung birgt aber auch das Risiko, dass dabei eingebrachte Fehler zu einem Fehlverhalten oder dem Ausfall eines Systems führen. Insbesondere bei sicherheitskritischen Anlagen ist das Testen nach Änderung der Steuerungssoftware daher unabdingbar. Eine besondere Herausforderung stellt dabei Testen von verteilten Software-Systemen dar, da oftmals komplexe Abhängigkeitsbeziehungen existieren. Da Funktionsänderungen an Steuerungssystemen zunehmend in der Betriebsphase notwendig werden, stellt dies hohe Anforderungen an Anlagenbetreiber. Da für Anlagenbetreiber in der Vergangenheit hauptsächlich prozesstechnische und maschinenbauliche Fragestellungen relevant waren, besitzen sie oftmals weniger Erfahrung im Bereich des Softwaretests.

Zur Unterstützung von Anlagenbetreibern bei der Absicherung von softwarebasierten Änderungen an Steuerungssystemen wird in der vorliegenden Arbeit ein Konzept für einen strukturierten und automatisierten Absicherungsprozess vorgestellt. Es zielt darauf ab, durch eine automatisierte Auswirkungsanalyse und Generierung formaler Verhaltensmodelle, formale Verifikationsmethoden für Anlagenbetreiber nutzbar zu machen.

Die automatisierte Generierung der zur Verifikation notwendigen Verhaltensmodelle basiert auf einem modularen Modellierungsansatz und Modelloperationen. Ausgehend von der Software-Änderung einer Steuerungskomponente werden, anhand einer Auswirkungsanalyse, die funktionalen Anforderungen identifiziert, deren Gültigkeit nach der Änderung nicht mehr gewährleistet ist. Für die betroffenen funktionalen Anforderungen wird anhand verschiedener Modelloperationen das zur Verifikation notwendige Verhaltensmodell generiert. Über formale Verifikationsmethoden wird anschließend überprüft, ob das generierte Verhaltensmodelle den spezifizierten funktionalen Anforderungen genügt.

Das Konzept wurde in Form des Testgeräts „TestIAS“ umgesetzt. Im Rahmen einer empirischen Evaluierung wurden an einem verteilten, serviceorientierten Automatisierungssystem neun Änderungsszenarien durchgeführt, welche durch TestIAS überprüft wurden. Damit konnte gezeigt werden, dass sich mithilfe des entwickelten Konzepts ein Absicherungsprozess umsetzen lässt, der für Anlagenbetreiber automatisiert durchführbar ist. Als Ergebnis dieses Prozesses stehen die Aussagen, welche Teilsysteme von einer Funktionsänderung betroffen sind und ob durch die Funktionsänderung funktionale Anforderungen verletzt wurden. Sind funktionale Anforderungen verletzt, wird der Anlagenbetreiber bei Eingrenzung der Fehlerursache unterstützt. Dazu benötigt der Anlagenbetreiber keinerlei Kenntnis über die Abhängigkeiten innerhalb des Automatisierungssystems. Dies geschieht, indem durch eine automatisierte Modellgenerierung die Vorteile funktionaler Verifikationsmethoden für Anlagenbetreiber nutzbar gemacht werden.

Abstract

Economic and technological drivers mean that control systems of automation systems are increasingly turning into complex software systems. Software enables the realization of complex control tasks and, due to its easy modifiability, a versatile control system. However, every modification also entails the risk that errors introduced in the modification process could lead to malfunction or failure of a system. Testing after modifying the control software is therefore indispensable, especially for safety-critical systems. Testing distributed software systems poses a particular challenge, as complex dependency relationships often exist. Since functional modifications to control systems are increasingly necessary in the operating phase, high demands are placed on production line operators. In the past, production line operators were mainly concerned with process engineering and mechanical engineering issues, so they often have less experience in software testing.

This paper presents a concept for a structured and automated verification process, to support production line operators in verifying software-based modifications to control systems. It aims at making formal verification methods usable for plant operators by an automated impact analysis and generation of formal behavior models.

The automated generation of the behavior models that are necessary for verification is based on a modular modeling approach and model operations. Starting from the software modification of a control component, the functional requirements of which the validity is no longer guaranteed after the modification are identified. This is realized on the basis of an impact analysis. The behavioral model that is necessary for verification of an affected functional requirement is generated by using various model operations. Then formal verification methods are used to check whether the generated behavior models meet the specified functional requirements.

The concept was implemented by a test device named "TestIAS". Within the framework of an empirical evaluation, nine modification scenarios were carried out on a distributed, service-oriented automation system. Those modifications were verified by TestIAS. This showed that the developed concept can be used to implement a verification process which can be automated for production line operators. As a result of this process, information is given on which subsystems are affected by a functional modification and whether functional requirements are violated by the functional modification. If functional requirements are violated, the plant operator is supported in localizing the cause of the error. For this purpose, the plant operator does not require any knowledge of the dependencies within the automation system. This is done by making the advantages of functional verification methods available to production line operators through automated model generation.

1 Einleitung und Motivation

1.1 Bedeutung von Software in der Automatisierungstechnik

Bei Automatisierungssystemen werden vermehrt Funktionalitäten in der Steuerungssoftware umgesetzt [13]–[15], was die Realisierung komplexer Automatisierungslösungen erlaubt. Dies führt aber wiederum zu einer Steigerung der Komplexität der Prozesse und somit der Steuerungssoftware [16]. Zur Handhabung dieser Komplexität und zur Erhöhung der Flexibilität werden Funktionalitäten eines Automatisierungssystems zunehmend auf „intelligente“ Komponenten verteilt [17], [18]. Diese intelligenten Komponenten enthalten eine integrierte Steuerung, welche Automatisierungsaufgaben mithilfe ihrer Sensoren und Aktoren selbstständig übernehmen kann [19].

Der Wandel von zentral gesteuerten Automatisierungssystemen zu stark vernetzten, verteilt gesteuerten Automatisierungssystemen bildet die Basis für neue Anwendungen und Geschäftsmodelle [20]. So ermöglicht die einfache Änderbarkeit von, insbesondere verteilten, Softwaresystemen eine schnelle Adaption der Steuerungssoftware eines Automatisierungssystems an geänderte Kundenanforderungen und Umgebungsbedingungen. Dies liegt unter anderem daran, dass bei verteilten Softwaresystemen Komponenten aufwandsarm integriert, entfernt oder über over-the-air Software-Updates geändert werden können. Aufgrund der aufwandsarmen Änderbarkeit von Software wird sie deutlich öfter angepasst als Hardware [21].

Diese Wandelbarkeit wird zunehmend bedeutsam, da zur Bedienung kurzfristiger Konsum-Trends eine hohe Produktvielfalt, kürzere Produktzyklen und dadurch eine hohe Volatilität der Produktionsvolumen gefordert sind [22]–[25]. Die Steigerung der Bedeutung von Software für Produktionssysteme beschreibt der Visionär und Unternehmer Elon Musk folgendermaßen: „Ein gutes Produktionssystem ist [sei] vor allem eine Frage der Software“ [26].

Durch das Zusammenspiel von Software und Hardware sowie Echtzeitanforderungen werden Steuerungen von Automatisierungssystemen zunehmend komplex [24], [27], [28]. So müssen harte Echtzeitanforderungen berücksichtigt, aber auch das globale Betriebsmanagement berücksichtigt werden [29]. Dies stellt besonders in der Automatisierungstechnik eine große Herausforderung dar, da oftmals sicherheitskritische Anwendungen realisiert werden. Dabei können einzelne Fehler in der Steuerungssoftware zum Ausfall einer Produktionsanlage oder gar zu einem gefährlichen Ereignis führen. Das deshalb geforderte niedrige Grenzrisiko stellt hohe Anforderungen an die Softwarequalität.

Zur Gewährleistung der Sicherheitsintegrität ist das Testen der Software unabdingbar [30]. Die wachsende Dominanz der Steuerungssoftware in der Automatisierungstechnik macht diese Absicherung zunehmend herausfordernd [31].

1.2 Herausforderungen bei der Absicherung von Software-Änderungen an verteilten Steuerungssystemen

Zur Gewährleistung einer gleichbleibend hohen Software-Qualität ist das Testen von geänderter Steuerungssoftware unabdingbar. Werden eine Software oder ihre Umgebung geändert, muss zum einen überprüft werden, ob die gewünschte Funktion erfüllt ist, und zum anderen, ob es ungewollte Änderungen oder Seiteneffekte gibt. Zur Reduktion des Testaufwands ist die Eingrenzung möglicher Seiteneffekte essentiell. Diese Eingrenzung kann anhand einer Auswirkungsanalyse durchgeführt werden, welche die Abhängigkeiten innerhalb eines Systems berücksichtigt [10].

Folgende Faktoren stellen für die Absicherung von Steuerungen zukünftiger Automatisierungssysteme eine große Herausforderung dar [32], [33]:

- die Vielzahl an Funktionalitäten von Steuerungen, die mit Software realisiert wird
- die hohe Individualität der Steuerungen von Automatisierungssystemen
- die komplexen Abhängigkeiten in dezentralen Steuerungssystemen, die für den Anlagenbetreiber oftmals unbekannt sind
- die Kooperation von Steuerungskomponenten verschiedener Hersteller
- die häufigen Funktionsänderungen durch Software-Updates führen dazu, dass Dokumentationen und Systemmodelle oftmals veraltet sind. Daher sind aktuelle Modelle vielfach nicht verfügbar.

In [32], [33] wird dies anhand theoretischer Betrachtungen hergeleitet und durch eine Expertenbefragung untermauert.

Die Vielzahl an Funktionalitäten industrieller Steuerungssysteme, welche durch Software realisiert ist, führt zu einem hohen Testaufwand. Im Gegensatz zu Verbraucherprodukten, welche teilweise auch über einen hohen Funktionsumfang verfügen, werden industrielle Automatisierungssysteme meist nach individuellen Anforderungen entwickelt, weshalb sie meist Einmal-Systeme sind [34]. Aufgrund dieser hohen Individualität ist es nicht möglich, die Auswirkungen von Funktionsänderungen, wie bei Verbraucherprodukten, ausgiebig im Labor zu testen und anschließend das Software-Update an alle Produkte dieses Typs ohne die Notwendigkeit weiterer Tests auszuliefern. So muss jedes Automatisierungssystem nach Änderungen separat getestet werden.

Bei Abarbeitung einer Automatisierungsaufgabe interagieren eine Vielzahl von Steuerungskomponenten verschiedener Hersteller, wodurch komplexe Abhängigkeiten zwischen den Softwarekomponenten entstehen. Diese Heterogenität führt dazu, dass dem Anlagenbetreiber die Abhängigkeiten innerhalb des Steuerungssystems oftmals unbekannt sind. Erschwerend kommt hinzu, dass sich Abhängigkeiten durch Integration, Entfernung oder Änderung von Komponenten verändern. Dies resultiert darin, dass das Pflegen von Systemmodellen eines Steuerungssystems hohe Expertise und hohen Aufwand erfordert und deshalb meist keine aktuellen Abhängigkeitsmodelle existieren.

Aus diesen Gründen werden heutzutage, gemäß dem allgemeinen Instandhaltungsgrundsatz „never touch a running system“, Änderungen an Produktionsanlagen gemieden [14]. Wie in Abbildung 1 dargestellt, spielt deshalb das Testen während der Betriebsphase eine untergeordnete Rolle. Während des Betriebs werden neben der Produktion hauptsächlich qualitätssichernde Maßnahmen (QS) betrieben. Das Testen ist heute hauptsächlich Bestandteil des Engineerings und der (Wieder-)Inbetriebnahme.

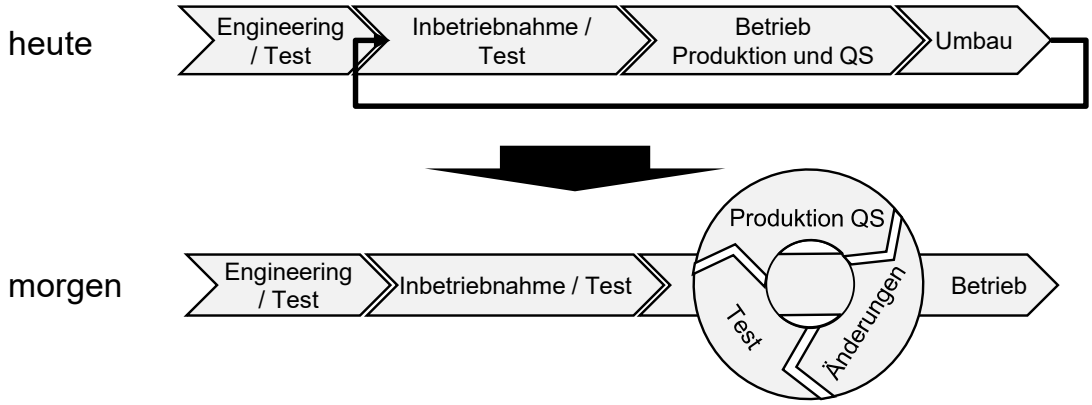


Abbildung 1: Änderung des Lebenszyklus einer Produktionsanlage [32]

Umbaumaßnahmen, zum Beispiel zur Produktion eines neuen Produkttyps, sind heute relativ selten und werden während der sich anschließenden Wiederinbetriebnahme abgesichert. Zukünftig wird vermehrt Änderungsbedarf während der Betriebsphase entstehen. Diese Funktionsänderungen müssen möglichst effizient durch Tests abgesichert werden. Da diese Änderungen während des Betriebs stattfinden sollen, stellt das langwierige Testen der Wiederinbetriebnahme keine Option dar. Dadurch entsteht während der Betriebsphase ein Kreislauf aus Produktion mit Qualitätssicherung (QS), Änderungen und Tests.

Dies führt dazu, dass auch das Testen von Software in der maschinenbaulich geprägten Domäne der Anlagenbetreiber zunehmend relevant wird. So ergab eine Expertenbefragung, dass, vor allem in der Betriebsphase, die Herausforderungen im Bereich Testen von Automatisierungssystemen in den nächsten zehn Jahren stark zunehmen werden [32]. Weitere Herausforderungen beim Testen im Betrieb stellt der hohe Informationsverlust über den Aufbau des Steuerungssystems dar, welcher sich meist im Übergang von der Entwicklungs- in die Betriebsphase einstellt.

Dabei fehlt es an effizienten Methoden, Werkzeugen und strukturierten Prozessen, um Anlagenbetreiber bei der Absicherung ihrer Produktionsanlagen nach Software-Änderungen zu unterstützen. Aus den beschriebenen Herausforderungen ergibt sich die zentreale Forschungsfrage der vorliegenden Untersuchung:

Wie können nach funktionalen Änderungen an der Steuerungssoftware verteilter Automatisierungssysteme automatisiert Fehler erkannt werden?

Das Ziel dieser Arbeit besteht in der Beantwortung dieser Frage. Dies wird im Folgenden näher erläutert.

1.3 Zielsetzung der Arbeit

Die Herausforderungen für Anlagenbetreiber bei der Absicherung von Funktionsänderungen an der Steuerungssoftware bilden den Ausgangspunkt für die vorliegende Arbeit. Aus der in Kapitel 1.2 hergeleiteten Forschungsfrage ergibt sich die Zielsetzung dieser Arbeit:

Entwurf eines Konzepts zur automatisierten Erkennung von Fehlern nach funktionalen Änderungen an der Steuerungssoftware.

Zur Sicherstellung der Anwendbarkeit der zu entwerfenden Lösung für Anlagenbetreiber soll das zu entwickelnde Konzept folgende Nebenbedingungen berücksichtigen:

1. Nebenbedingung: Berücksichtigung der Rahmenbedingungen zukünftiger Steuerungen von Automatisierungssystemen

Im Zuge der Digitalisierung und Vernetzung industrieller Automatisierungssysteme entsteht ein struktureller Wandel hin zu verteilten, wandelbaren Steuerungssystemen. Die Herausforderungen und Möglichkeiten dieser neuen Systemstruktur, auf die in Kapitel 2.1 eingegangen wird, soll Berücksichtigung finden.

2. Nebenbedingung: Einfache Integration in Automatisierungssysteme

Greift das Konzept zu stark in die Struktur oder das Verhalten bestehender Automatisierungssysteme ein, besteht die Gefahr, dass der Integrationsaufwand den potenziellen Nutzen übersteigt. Somit hat die Komplexität der Inbetriebnahme einen bedeutenden Einfluss auf die Nutzbarkeit eines Konzeptes.

3. Nebenbedingung: Anwendbarkeit in der Betriebsphase

Over-the-air Software-Updates und Ad-hoc-Vernetzung erleichtern die Änderung von Steuerungssystemen. Diese Änderungen werden typischerweise in der Betriebsphase stattfinden, weshalb ein wichtiger Aspekt der Arbeit darin besteht, die Anwendbarkeit des zu entwickelnden Konzepts in dieser Phase sicherzustellen.

4. Nebenbedingung: Hohe Effizienz

Verfahren werden nur dann genutzt, wenn ein Mehrwert entsteht. Dieser Mehrwert hängt entscheidend von der Effizienz des Verfahrens ab. Wichtige Faktoren sind der Automatisierungsgrad und die notwendige Zeitdauer der Fehlererkennung.

5. Nebenbedingung: Einfache Nachvollziehbarkeit

Neben der Effizienz ist die einfache Nachvollziehbarkeit ein wichtiger Faktor, um Mehrwert zu generieren. Dies beinhaltet eine klare Darstellung des Ergebnisses der Absicherung. Im Fall eines Fehlers soll deutlich werden was für ein Fehler vorliegt und das Ergebnis bei der Eingrenzung der Fehlerquelle hilfreich sein.

1.4 Gliederung der Arbeit

Die vorliegende Arbeit ist in sieben Kapitel gegliedert. In Kapitel 2 werden die für die Arbeit notwendigen Grundlagen beschrieben und der Stand der Forschung beleuchtet. Daraus wird der Forschungsbedarf abgeleitet. Auf Basis des dargestellten Forschungsbedarfs wird in Kapitel 3 das Konzept der Absicherung verteilter Automatisierungssysteme vorgestellt. Dazu werden zuerst der komponentenbasierte Modellierungsansatz und die Verhaltensmodellierung präsentiert, welche die Grundlage für das Konzept darstellen. Darauf aufbauend wird der dreistufige, automatisierte Absicherungsprozess erläutert.

Im darauffolgenden Kapitel 4 wird die Umsetzung des Konzepts anhand des Prototyps TestIAS beschrieben. Zur Demonstration der Anwendbarkeit und Evaluierung des Konzepts wird in Kapitel 5 ein verteiltes Automatisierungssystem als Testobjekt eingeführt, an welchem sich Funktionsänderungen der Steuerungssoftware durchführen lassen. Die dazu entworfene Steuerung weist typische Eigenschaften von Steuerungen zukünftiger Automatisierungssysteme auf.

Schließlich wird in Kapitel 6 die Evaluierung des Konzepts beschrieben. Dabei bildet TestIAS die Basis für die empirischen Untersuchungen. Hierzu werden verschiedene Änderungsszenarien für das verteilte Steuerungssystem beschrieben. Die von TestIAS berechneten Ergebnisse des Absicherungsprozesses nach Durchführung der Änderungen werden aufgezeigt und interpretiert.

Zum Abschluss werden in Kapitel 7 die wichtigsten Aspekte der Arbeit zusammengefasst. Des Weiteren werden Ergebnisse aufgeführt und Möglichkeiten zur Weiterführung der Arbeit aufgelistet.

2 Stand der Technik und Forschung

Dieses Kapitel führt in den Stand der Technik und Forschung im Bereich der Anlagenautomatisierung und im Bereich des Softwaretests von Steuerungssystemen ein. Nach Darlegung der Grundlagen wird aktuelle Literatur diskutiert und Forschungsbedarf aufgezeigt.

2.1 Zukunft der Anlagenautomatisierung

Im Kontext der Digitalisierung und Vernetzung der Produktion findet ein elementarer Wandel der Automatisierungstechnik statt. Die Verzahnung der Produktion mit Kommunikations- und Informationstechnik soll die Konkurrenzfähigkeit des Produktionsstandorts Deutschland erhalten [35]. Die Ursachen und die zu erwartenden technischen Auswirkungen auf Steuerungssysteme werden im Folgenden betrachtet. Die Betrachtung beschränkt sich auf konzeptrelevante Faktoren.

2.1.1 Treiber des Wandels

Volatile Marktanforderungen, flexible Kundenwünsche und der weltweite Wettbewerb sind wirtschaftliche Treiber, welche eine wandlungsfähige Produktion erfordern [36], [37]. Zur Bedienung dieser Anforderungen sind technologische Treiber notwendig, die eine wandlungsfähige, wirtschaftliche Produktion ermöglichen. Nach [38] gibt es folgende technologische Treiber der Industrie 4.0:

- **Cyber-physische (Produktions-)Systeme (CPPS)** beschreiben vernetzte Komponenten, die über Sensorik und Aktorik verfügen, fortgehend Daten austauschen und automatisiert mit einem „intelligenten Produkt“ die Produktion planen und steuern. Intelligentes Produkt beschreibt dabei eine aktive Komponente, die ihren Bauplan kennt und den Produktionsprozesses steuert und überwacht [38].
- **Integrierte Daten, Datenströme und Big Data** basieren auf der Vernetzung von Komponenten. Durch die sogenannte horizontale und vertikale Integration werden eine Vielzahl an Daten unabhängig von Automatisierungsebenen und Unternehmensgrenzen ausgetauscht. Diese werden gesammelt, analysiert und interpretiert [38].
- **5G und Cloud-Technologien** ermöglichen die weltweite Vernetzung mit hoher Bandbreite sowie die massenhafte Speicherung, Verwaltung und Analyse von Prozessdaten [38].
- **Additive Fertigungsverfahren** erlauben höchst flexible Produktionsprozesse, bei welchen Produkte mittels 3D-Druck aus Materialen wie Kunststoffe, Verbundwerkstoffe und Metalle kundenindividuell und automatisiert hergestellt werden können [38].

2.1.2 Steuerungsstrukturen der Automatisierungstechnik im Wandel

Aufgrund der im vorigen Kapitel beschriebenen wirtschaftlichen und technologischen Treiber ändert sich die Struktur von Steuerungssystemen fundamental. Bei der Realisierung von wandlungsfähigen Produktionen sind folgende Tendenzen zu beobachten:

- modularer und verteilter Steuerungsstrukturen [35]
- vielfältige Vernetzung zwischen den Komponenten aufgrund der hohen Kommunikationsfähigkeit der Komponenten [35]
- hoher Grad an Software-Änderbarkeit [24]

Nach Industrie-4.0-Visionen bilden Cyber-physische Produktionssysteme die technologische Basis einer wandlungsfähigen Produktion [39]. Werden klassische Prozesssteuerungen meist durch eine zentrale Steuerungseinheit koordiniert (siehe Abbildung 2 links), zeichnen sich Cyber-physische Produktionssysteme durch eine verteilte Steuerungsstruktur aus (siehe Abbildung 2 rechts) [40]. In einer verteilten Steuerungsstruktur ist die Funktionalität auf mehrere Komponenten über Rechnergrenzen hinaus verteilt [12]. Das verbreitetste Modell verteilter Systeme ist das Client-Server-Modell, bei welchem ein Client die Dienstleistung eines Servers (Service) in Anspruch nehmen kann [12]. Bei dieser Server-Client Beziehung existieren eine logische Hierarchie und dadurch eine Abhängigkeitsbeziehung zwischen den Komponenten – ein Client ist abhängig von der Funktionalität eines Servers.

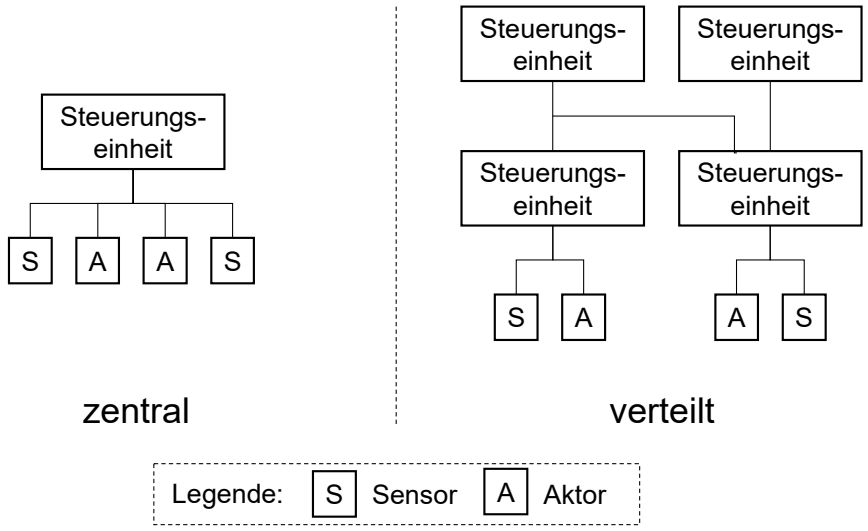


Abbildung 2: Gegenüberstellung einer zentralen Steuerungsstruktur und einer verteilten Steuerungsstruktur

Weitere mögliche Eigenschaften verteilter Komponenten, wie Ad-hoc-Vernetzbarkeit, Selbstkonfigurierbarkeit und umfangreiche Interaktionsfähigkeit [37], ermöglichen eine Ablösung fest programmierter, zentraler Prozesssteuerungen durch dynamische, dezentrale Prozesssteuerun-

gen [17]. Die verteilte Steuerungsstruktur und die umfangreichen Kommunikations- und Interaktionsfähigkeiten der Komponenten erlauben somit einen hohen Grad an Änderbarkeit der Produktionssysteme [41], [42]. Dies wird zunehmend relevant, um Steuerungssysteme an geänderte Umgebungsbedingungen und Auftragsanforderungen anzupassen [43]. Die Ad-hoc-Vernetzbarkeit und lose Kopplung der Komponenten von Cyber-physischen Systemen ermöglichen eine Skalierung des Steuerungssystems ohne hohen Konfigurationsaufwand. Zur Umsetzung einer flexiblen und verteilten Prozesssteuerung ist ein durchgehender Informationsfluss innerhalb des Systems unabdingbar [35], [41]. Dies setzt eine hohe Maschine-zu-Maschine-Kommunikationsfähigkeit (M2M) der Komponenten voraus. Diese M2M-Kommunikationsfähigkeit der Komponenten erlaubt es darüber hinaus Funktionsänderungen an der Steuerungssoftware über over-the-air Software-Updates aufwandsarm durchzuführen.

Die verteilte flexible Prozesssteuerung resultiert in komplexen Abhängigkeiten zwischen Komponenten, welche bei Abarbeiten eines Steuerungsprogramms entstehen [44]. Zur Koordination dieser verteilten, wandlungsfähigen Steuerungssysteme werden flexible Architekturen benötigt. Dafür eignen sich serviceorientierte Architekturen (SOA), welche im Folgenden vorgestellt werden.

2.1.3 Serviceorientierung in der Automatisierungstechnik

Serviceorientierte Architekturen (SOA) sind eine Basisarchitektur für verteilte informationstechnische Systeme. Sie dienen dazu, Services (deutsch: Dienste) innerhalb eines Netzwerks zu verwalten und zu nutzen [45]. Aufgrund ihrer Eignung für wandlungsfähige Informationssysteme werden sie vermehrt als das M2M-Kommunikationsparadigma zukünftiger Steuerungen für Automatisierungssysteme angesehen [11]. Eine SOA ist gekennzeichnet durch folgende Eigenschaften [45]:

- verteiltes System
- standardisiert zugreifbare Serviceschnittstellen
- lose Kopplung zwischen den Komponenten
- Wiederverwendbarkeit der Komponenten

Eine SOA besteht aus Komponenten, die miteinander in einer Client-Server-Beziehung stehen. Server bieten Fähigkeiten an, welche in Services gekapselt sind und von Clients genutzt werden können. Die Fähigkeiten eines Servers sind über semantisch beschriebene, standardisiert zugreifbare Service-Schnittstellen aufrufbar [45], [46]. Kapselung beschreibt die Trennung von Schnittstelle und Implementierung. Dies erlaubt eine technologieunabhängige Implementierung der Funktionalität und somit die Kooperation heterogener Technologien in einem System [47]. Durch

offene Standards können somit Komponenten verschiedener Hersteller und Domänen in ein System integriert werden. So ist die SOA die Schlüsseltechnologie um eine Integration verschiedener Ebenen der Automatisierungspyramide nach dem Standard ANSI/ISA-95 [48] zu realisieren [29].

Zur Ausführung komplexer Produktionsprozesse können höherwertige Services durch Kombination verschiedener Services zur Laufzeit zusammengestellt werden. Dieses Vorgehen wird Orchestrierung genannt. Auf Basis der modularen Service-Struktur und der Wiederverwendbarkeit der Services erlaubt die Orchestrierung eine sehr flexible und zügige Erstellung von höherwertigen Services [49]. Zur Ausführung eines höherwertigen Service muss der Nutzer kein detailliertes Wissen über den Ablauf des Prozesses besitzen [46], [50]. Diese hierarchische Modularisierung erlaubt es, komplexe Gesamtprozesse durch das Zusammenspiel einzelner, weniger komplexer Teilprozesse zu realisieren [51].

Die lose Kopplung zwischen den Teilnehmern erleichtert die Integration neuer Teilnehmer ins Netzwerk [52]. Lose Kopplung beschreibt den Sachverhalt, dass Services durch einen Client bei Bedarf gesucht, gefunden und dynamisch gebunden werden können. Eine dynamische Bindung findet dabei zwischen dem Client und dem Server, der einen Service anbietet, zur Laufzeit statt. Dies erlaubt es, ohne Konfigurationsaufwand neue Netzwerkteilnehmer hinzuzufügen (Ad-hoc-Integration) und somit die Skalierung eines Systems zur Laufzeit [50]. Meist findet bei der losen Kopplung die Kommunikation asynchron statt [46]. Dies hat den Vorteil, dass ein Teilnehmer nach Funktionsaufruf nicht blockiert ist, bis er eine Antwort erhält. Es stellt aber höhere Anforderungen an die Kommunikation, besonders wenn Antworten mehrerer Funktionsaufrufe in unterschiedlicher Reihenfolge eintreffen können [46].

Der Mechanismus der losen Kopplung ist prinzipiell in Abbildung 3 dargestellt. Damit ein Service für Clients sichtbar wird, registriert jeder Server im Netzwerk seine Services bei einem Discovery-Server, welcher ein Serviceverzeichnis führt. Semantische Beschreibungen der Services garantieren, dass Netzwerk-Teilnehmer die Servicebeschreibungen korrekt interpretieren können. Bei Bedarf können Clients beim Discovery-Server anfragen, welche Server geeignete Services anbieten. Dieser übermittelt eine Liste mit den Referenzen der Server. Unter den geeigneten Services obliegt es dem Client zu entscheiden, welcher Server zur Ausführung des Service aufgerufen werden soll. Anschließend folgt die Bindung zwischen Client und Server, über welche weitere Daten angefragt und Serviceaufrufe durchgeführt werden können [47].

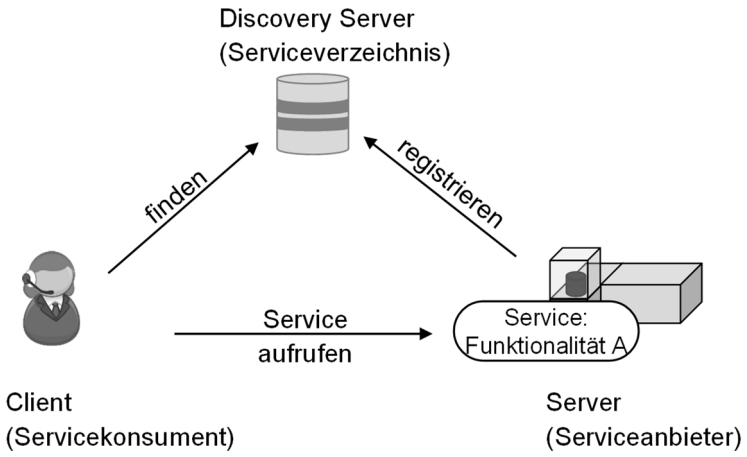


Abbildung 3: Prinzip der losen Kopplung einer SOA

Zur Realisierung einer hierarchischen Orchestrierung über mehrere Ebenen müssen Services ihrerseits die Möglichkeit besitzen, weitere Services aufzurufen. Deshalb benötigen diese Netzwerkteilnehmer neben einem Server auch einen Client, welcher unter anderem Serviceaufrufe und den Discovery-Prozess ermöglicht. Eine geeignete Realisierung ist in Abbildung 4 dargestellt.

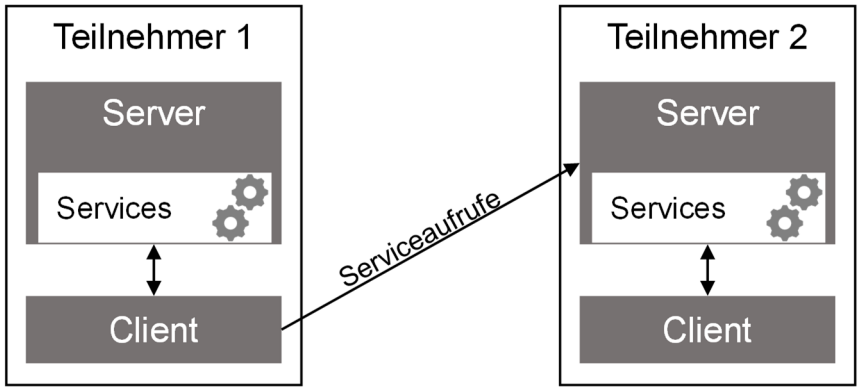


Abbildung 4: Möglicher Aufbau und Aufruf von Netzwerk-Teilnehmern in verteilten Systemen nach [53].

Aufgrund dieser Eigenschaften, welche einen flexiblen und fehlertoleranten Betrieb ermöglichen [50], eignen sich serviceorientierte Architekturen sehr gut zur Umsetzung des in Kapitel 2.1.2 beschriebenen strukturellen Wandels von Steuerungssystemen. So werden serviceorientierte Architekturen als ein wahrscheinliches M2M-Kommunikationsparadigma zukünftiger Steuerungssysteme angesehen. Relevante serviceorientierte Protokolle sind hierbei OPC UA, MTConnect und MQTT. Diese offenen Standards ermöglichen eine semantische Beschreibung der Daten und setzen auf standardisierte Netzwerkprotokolle wie etwa TCP/IP [17].

Dabei wird der Begriff „Service“ nicht einheitlich verwendet. In der Automatisierungstechnik wird ein Service als „abgegrenzter Funktionsumfang, der von einer Entität oder Organisation über Schnittstellen angeboten wird“ definiert [11]. Dagegen wird in der Domäne Informatik als Service

der Mechanismus beschrieben, der Bedürfnisse und Fähigkeiten von Softwarekomponenten zusammenbringt [54]. In [55] werden dazu vier verschiedene Servicehierarchien unterschieden. „Communication Services“ beschreiben einfache Fähigkeiten, die zum Datentransfer notwendig sind, wie beispielsweise für den Verbindungsaufbau. „Information Services“ ermöglichen einfache Lese- und Schreibzugriffe auf das Datenmodell eines Servers. Darauf aufbauend erlauben „Plattform Services“ Selbstmanagementfunktionen. Die oberste Hierarchieebene bilden „Application Services“, welche die Fähigkeiten von Produktionssystemen beschreiben. In dieser Arbeit wird der Begriff „Services“ für „Application Services“ verwendet.

Als Alternative zur SOA können Multi-Agenten-Systeme angesehen werden, welche ähnliche Eigenschaften besitzen [53]. Eine weitergehende Eigenschaft von Agenten ist, dass sie nicht nur auf die Umwelt reagieren, sondern proaktiv mit dieser interagieren. Um diese proaktive Interaktion zu realisieren, verfügen Agenten über Intelligenz. Aufgrund dieser Eigenschaften werden Multi-Agenten-Systeme oftmals für übergeordnete Optimierungsaufgaben verwendet werden. Dagegen können SOA auch zur maschinennahen Kommunikation, wie der Prozessdatenerfassung und Prozesssteuerung, verwendet werden. Ein Ansatz, wie Multi-Agenten-Systeme und SOA sinnvoll in einem System kombiniert werden können, wird in [56] vorgestellt. Für eine Beschreibung und realisierte Anwendungen von Multi-Agenten-Systemen wird auf [43], [57]–[59] verwiesen.

2.2 Modelle zur Beschreibung des Verhaltens von Automatisierungssystemen

Die Schaffung eines für alle Beteiligten gemeinsamen Verständnisses über den Aufbau eines Automatisierungssystems stellt eine große Herausforderung dar [60]. Zur Erreichung eines einheitlichen Verständnisses werden zum Entwurf von Verhalten und Struktur komplexer Automatisierungssysteme üblicherweise Modelle verwendet. Somit sind Modelle beim Engineering und Betrieb von Automatisierungssystemen von zentraler Bedeutung.

Nach Stachowiak zeichnen sich Modelle durch folgende Hauptmerkmale aus [61], [62]:

- Abbildung
- Abstraktion
- Pragmatik (Modellierungszweck)

Demnach sind Modelle vereinfachende Abbildungen von Realitäten unter Betrachtung einer bestimmten Sicht. Die Verwendung einer beschränkenden Sicht auf die Realität erlaubt durch Abstraktion das Hervorheben der für den Modellierungszweck relevanten Aspekte [63]. Im Rahmen einer Modellbildung werden Modelle durch Analyse und Abstraktion einer Realität erstellt [64]. Bei der Realität kann es sich um ein bereits existierendes (deskriptives Modell) oder ein zu entwerfendes System (präskriptives Modell) handeln [65].

Modelle besitzen zwei Hauptaspekte – semantische Information und visuelle Präsentation. Unter semantischer Information versteht man die logische Formulierung von Zuständen, Zustandsübergängen, Klassen oder Nachrichten. Sie beschreibt die Struktur eines Modells und wird zur maschinellen Verarbeitung der Modelle benötigt. Eine visuelle Präsentation wird von den meisten PC-Programmen nicht benötigt, erhöht jedoch das menschliche Verständnis. Sie erlaubt es Menschen, auf verständliche Art und Weise semantische Informationen einzusehen, zu suchen oder zu editieren [63]. Abhängig von dem Zweck eines Modells können verschiedene Aspekte eines Systems betrachtet werden. Typische Modellierungszwecke sind die Schaffung eines gemeinsamen Verständnisses, der Entwurf von Systemen, die Organisation und Strukturierung großer Systeme und die Analyse und Simulation komplexer Systeme [63].

In der Automatisierungstechnik steigt die Bedeutung von Modellen in sämtlichen Lebenszyklusphasen eines Automatisierungssystems. Dabei hilft die Abstraktion, die zunehmende Größe und Komplexität von Automatisierungssystemen beherrschbar zu machen. Typische Anwendungsfälle sind:

- Bei der modellbasierten Entwicklung werden Modelle zur Beschreibung eines zu entwerfenden Systems verwendet.
- Beim modellbasierten Test werden Modelle verwendet, um Testartefakte anschaulich darzustellen und Schritte des Testprozesses zu automatisieren.
- Im Kontext des digitalen Zwillings werden Informationen über den Lebenszyklus eines Objekts gesammelt, um ein virtuelles Abbild des realen Objekts zu erhalten. Modelle werden dabei zur Strukturierung oder Verknüpfung von Information verwendet.
- Die virtuelle Inbetriebnahme ermöglicht die Reduktion der Inbetriebnahmezeit durch eine vorab durchgeführte Simulation des jeweiligen Systems. Diese Simulationen basieren auf ausführbaren Modellen.

In den meisten der aufgelisteten Anwendungsfälle werden Modelle benötigt, die das Verhalten oder die Struktur von Automatisierungssystemen beschreiben. Diese Modelle entstehen meist bei der modellbasierten oder modellgetriebenen Entwicklung und können für Simulationszwecke bei der Entwicklung, der Inbetriebnahme oder während des Betriebs genutzt werden. Austauschformate erlauben die Portierung von Modellinformation über den Lebenszyklus [66]. Eine gesamtheitliche Beschreibung von Automatisierungssystemen ermöglicht die Vernetzung von Information über Domänen hinweg. Dazu werden multidisziplinäre Modellierungsansätze wie SysML, Modelica [67] oder die formalisierte Prozessbeschreibung benötigt [60].

Da die vorliegende Untersuchung auf Software-Änderungen fokussiert, wird im Folgenden die Modellierung von diskreten Softwaresystemen beleuchtet.

2.2.1 Modellierung der Software von Steuerungssystemen

Die verbreitetste Software-Modellierungssprache ist die Unified Modeling Language (UML). Zur Beschreibung von Software definiert die UML die Semantik und visuelle Präsentation von 14 Modelltypen (Version: UML 2.4.1). UML, welche durch die Object Management Group (OMG) standardisiert wurde, zielt darauf ab, den objektorientierten Entwurf von ereignisdiskreten Softwaresystemen zu strukturieren. Dabei sind für jede Entwicklungsphase geeignete Modelltypen vorgesehen. Durch den Einsatz von UML findet im Entwicklungsprozess kein Medienbruch bei der Modellierung statt. Somit kann Modellinformationen in die nächsten Entwicklungsphasen übernommen und dort verfeinert werden [63].

Im Bereich der Systementwicklung wurde die auf UML basierende Modellierungssprache SysML durch die OMG standardisiert. Da zur Entwicklung komplexer technischer Systeme andere Beschreibungsmittel notwendig sind, wurden einige Modellierungsarten der UML modifiziert und zusätzliche Modellierungsarten hinzugefügt [68].

In [69] wird ein formalisierter Software- und Systementwicklungsprozess für verteilte, reaktive Systeme (FOCUS) vorgestellt. Dieser Prozess beschreibt, wie man mit formalen, hierarchischen Modellen durch Schnittstellenabstraktion Systeme entwickelt. Dies basiert auf einer schrittweisen Verfeinerung der Modelle während des Entwicklungsprozesses. Dabei sollen aufgrund der formalen Darstellung sämtliche Hierarchiestufen verifizierbar sein. Wegen der hohen Komplexität stellt dieser mathematisch formale Ansatz eine hohe Hürde für den praktischen industriellen Einsatz dar [70].

Die beschriebenen modellbasierten Vorgehensweisen erlauben eine domänenübergreifende Betrachtung von Automatisierungssystemen. Angesichts der Zielsetzung der vorliegenden Arbeit werden nachfolgend Modellierungsarten vorgestellt, welche das Verhalten und die Abhängigkeiten verteilter Softwaresysteme darstellen.

2.2.1.1 Verhaltensmodellierung verteilter Steuerungen von Automatisierungssystemen

Die Wahl einer geeigneten Modellierungsart kann entscheidend für den Erfolg einer Methode sein. Aus diesem Grund werden folgend Verhaltensmodelle für Steuerungen von diskontinuierlichen Prozessen (Folge- und Stückprozesse) betrachtet. Diese Steuer- und Ablaufmodelle sind durch diskrete, meist binäre, Prozessgrößen gekennzeichnet [64]. Sie bestehen aus einer Menge von Zuständen oder Aktionen, welche meist deterministisch voneinander abhängen. Die Übergänge zwischen den Zuständen oder Aktionen können zeitgesteuert oder ereignisgesteuert erfolgen. Zur Erzeugung von ausführbaren Programmcode aus einem Verhaltensmodells oder zur automatisierten Analyse muss ein Modell maschineninterpretierbar sein. Dazu bedarf es einer formal definierten Notation der Modellierungsart. Die modellbasierte Entwicklung zum Entwurf der formalen Modelle ist in der Automatisierungstechnik seit Jahrzehnten etabliert.

In Folge der zunehmenden Vernetzung und des Wandels zu serviceorientierten Architekturen gewinnen weitere Modellierungsarten an Relevanz. Da bei Geschäftsprozessen und Webtechnologien der Informatik ähnliche verteilte IT-Strukturen existieren, werden Modellierungsarten dieser Domänen, neben jenen der Automatisierungstechnik, folgend betrachtet.

Automaten

Die Grundform zur Beschreibung ereignisdiskreter Systeme sind Automaten. Ein Automat besteht aus einer definierten Zustandsmenge und einer Zustandsübergangsfunktion. Abhängig von der Eingabe kann es zu einem Übergang des Automaten von dem aktuellen Zustand in einen Folgezustand kommen. Zur Modellierung von Programmcode eignen sich deterministische, endliche Automaten, bei denen einem Zustand, bei gegebener Eingabe, eindeutig ein Folgezustand zugeordnet ist. Automaten lassen sich anhand gerichteter Graphen visualisieren. Bei diesen sogenannten Automatengraphen stellen Knoten die Zustände und Kanten die Zustandsübergänge dar (siehe Abbildung 5 a)) [71].

Vor allem in der Informatik werden Zustandsautomaten zur Modellierung von ausführbarem Programmcode eingesetzt [72]. So sind Zustandsautomaten unter anderem in UML und SysML spezifiziert. Sie ermöglichen eine eingängige Darstellung sequentieller Abläufe. Für eine detaillierte Beschreibung sei auf [71] verwiesen.

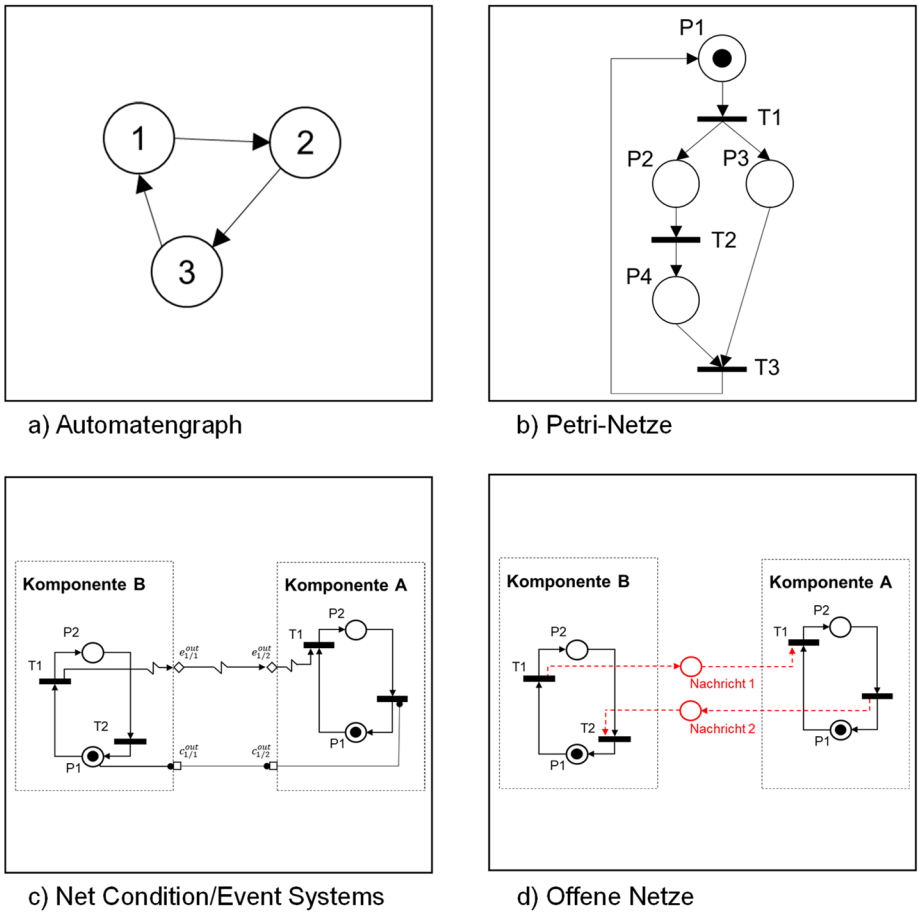
Petri-Netze

Petri-Netze stellen eine Erweiterung der Automaten dar, welche die Modellierung paralleler Prozesse ermöglicht. Dies wird durch Ersetzen eines globalen Zustands durch eine Kombination lokaler Zustände ermöglicht. Neben den Automaten sind Petri-Netze eine etablierte Modellierungsart für ereignisdiskrete Systeme. Ein Petri-Netz besteht aus zwei Knotentypen – den Stellen P und den Transitionen T. Wie in Abbildung 5 b) verdeutlicht, werden Stellen, welche einen lokalen Zustand repräsentieren, durch Kreise und Transitionen, welche Ereignisse repräsentieren, durch Rechtecke symbolisiert. Diese sind über gerichtete Kanten (Flussrelationen) verbunden, wobei nur Verknüpfungen zwischen Stellen und Transitionen zulässig sind. Token, dargestellt als schwarze Punkte, definieren über die Belegung von Stellen den Zustand des Systems. Durch Aufspaltung und Zusammenführung von Token an Transitionen können Parallelitäten dargestellt werden. Aufgrund der Beschreibbarkeit durch lineare algebraische Gleichungen können Petri-Netze, außer zum Systementwurf, zur Analyse des Systemverhaltens und der Performance verwendet werden [73]. So lassen sich Petri-Netze auf Verklemmung, Lebendigkeit, Beschränktheit und Sicherheit analysieren [72]. Auf diese Eigenschaften wird nachfolgend eingegangen:

Partielle Verklemmung beschreibt den Sachverhalt, dass die Zustände eines Petri-Netzes nur noch einen begrenzten Teil des ursprünglichen Zustandsraum erreichen können. Bei der totalen Verklemmung sind überhaupt keine Zustandsübergänge mehr möglich. Ein Petri-Netz ist lebendig, wenn es ausschließlich aus lebendigen Transitionen besteht. Das bedeutet, wenn in endlicher Zeit

jede Transition schaltbereit werden kann. Ein Petri-Netz ist beschränkt, wenn die Anzahl Token nicht unbegrenzt zunehmen kann. Die Sicherheit stellt eine Sonderform der Beschränktheit dar. Zur Erfüllung dieses Sachverhalts darf das Petri-Netz jederzeit nur ein Token besitzen.

Zur kompakteren Darstellung von Petri-Netzen wurden High-Level-Petri-Netze definiert und in der Norm ISO/IEC 15909 standardisiert [74]. High-Level-Petri-Netze erlauben die Definition zusätzlicher Attribute. So kann bei farbigen Petri-Netzen, durch sogenannte farbige Token, zusätzliche Information modelliert werden. Diese Information wird an die farbigen Token geheftet und kann verwendet werden, um das Verhalten des Petri-Netze zu beeinflussen. So können beispielsweise „guards“ in Transitionen integriert werden, welche das Schalten der Transitionen nur erlauben, wenn ein Token eine bestimmte Farbe besitzt. Dadurch können ähnliche Strukturen eines Petri-Netzes effizient zusammengefasst und Datenstrukturen modelliert werden. Für weitergehende Beschreibungen seid auf [75] verwiesen. Darüber hinaus beschreibt die Norm ISO/IEC 15909 Teil 2 das XML-basierte Austauschformat PNML (Petri Net Markup Language) für High-Level-Petri-Netze [76].



a) Automatengraph b) Petri-Netze

c) Net Condition/Event Systems d) Offene Netze

Abbildung 5: Verhaltensmodelle zur Beschreibung diskreter Prozesse der Automatisierungstechnik und Informatik

Petri-Netze sind universelle Beschreibungsmittel. Der universelle Charakter der Petri-Netze erlaubt die Anwendung in verschiedenen Bereichen, wobei die Bedeutung der Stellen und Transitionen unterschiedlich interpretiert wird [72]. Im Bereich der Automatisierungstechnik wurden 1977 erstmalig die steuerungstechnisch interpretierten Petri-Netze (SIPN) beschrieben [77]. SIPN, welche zur Projektierung von Speicherprogrammierbaren Steuerungen (SPS) geeignet sind, ermöglichen die Beschreibung von Schnittstellen der Steuerung mit dem technischen Prozess. Zur Modellierung des technischen Prozesses wurden analog dazu prozessbezogen interpretierte Petri-Netze (PIPN) entworfen. Für eine detaillierte Beschreibung wird auf [78] verweisen.

An SIPN wurde die Modellierungsart GRAFCET (Graphe fonctionnel de commande etapes/transitions) angelehnt. GAFCET wird unter anderem zur Darstellung ereignisdiskreter Systeme bei der objektorientierten Modellierungssprache Modelica eingesetzt [79]. GRAFCET wird ihrerseits als Vorgänger der in IEC 61131-3 standardisierten SPS-Programmiersprache SFC (Sequential Function Chart) angesehen [72].

Zur Modellierung modularer Steuerungssoftware wurden NCES (Net Condition / Event Systems) entwickelt [80]. NCES, welche in Abbildung 5 c) dargestellt sind, basieren auf typischen Petri-Netzen. Um Schnittstellen zwischen den Komponenten zu realisieren, wurden NCES durch Event- und Condition-Signale erweitert. Diese Signale verbinden, entgegen der klassischen Petri-Netz-Konvention, Stellen mit Stellen beziehungsweise Transitionen mit Transitionen. Somit stellen die bei NCES eingesetzten Flussrelationen eine Sonderform dar.

Zur Modellierung verteilter Softwaresysteme ist in der Informatik das Konzept des „offenen Netzes“ entstanden. Offene Netze, welche in Abbildung 5 d) dargestellt sind, basieren auf Petri-Netzen, welche um IT-Schnittstellen zur asynchronen Kommunikation erweitert wurden [81]. Sie finden unter anderem Verwendung in der Überprüfung der Kompatibilität von überorganisatorischen Prozessen. Bei der Modellierung serviceorientierter Architekturen kapseln diese „Interface-Stellen“, welche die asynchrone Kommunikation abbilden, die Funktionalität einer Komponente nach außen ab. In Abbildung 5 d) stellen *Nachricht 1* und *Nachricht 2* „Interface-Stellen“ dar. Über diese Schnittstellen lassen sich, analog zu NCES, die Petri-Netze verschiedener Komponenten komponieren. Im Gegensatz zu NCES wird die Konvention klassischer Petri-Netz eingehalten. Schnittstellen zum technischen Prozess werden bei offenen Netzen nicht berücksichtigt.

SPS-Programmiersprachen

Zur Projektierung von Speicherprogrammierbaren Steuerungen (SPS) wurden zahlreiche graphische und textuelle Sprachen zur Beschreibung des Verhaltens von Steuerungen entwickelt, welche in IEC 61131-3 standardisiert sind. Aufgrund fehlender IT-Schnittstellen und somit nur bedingter Eignung für verteilte Software-Systeme wird hier lediglich auf den IEC Standard verwiesen [82].

Geschäftsprozesse

Zur Beschreibung von Geschäftsprozessen existiert die graphbasierte Modellierungsart BPMN (Business Process Model and Notation), welche von der OMG standardisiert wurde [83]. BPMN soll eine für alle Beteiligten der Domäne Prozessmanagement nachvollziehbare Notation bereitstellen. Seit der Version 2.0 existiert eine formale Notation und ein standardisiertes XML-basiertes Austauschformat. Zur Ausführung des modellierten BPMN-Ablaufdiagramms kann dieses in die XML-basierte Sprache BPEL (Web Services Business Process Execution Language) umgewandelt werden [84], wobei die Ausdrucksfähigkeit nicht deckungsgleich ist. BPEL wurde von der Organisation OASIS standardisiert und stellt die Koordination zwischen Web-Services formalisiert dar. Neben BPMN werden EPK (ereignisgesteuerte Prozessketten) zur Geschäftsprozessmodellierung verwendet, welche im Rahmen des ARIS-Konzepts definiert sind. EPK erlauben die Verknüpfung von Geschäftsprozesse über „und“, „oder“ und „exklusiv-oder“ Attribute. Die erweiterte EPK (eEPK) erlaubt zusätzlich die Modellierung von Organisation, Daten und Leistung [85].

Zusammenfassende Bewertung

Die vorgestellten graphbasierten Modellierungsätze können anhand der in Tabelle 1 dargestellten Kriterien bewertet werden. Die Kriterien beschreiben relevante Eigenschaften zur Modellierung des Verhaltens verteilter Steuerungssysteme.

Modelle aus dem Bereich der Geschäftsprozesse beschreiben zwar verteilte Systeme, eignen sich aber aufgrund teilweise fehlender Formalisierung und der Vernachlässigung zeitlicher Aspekte nicht zur Modellierung verteilter Steuerungen der Automatisierungstechnik.

Zustandsautomaten sind weit verbreitet, formal und einfach nachvollziehbar. Aufgrund des hohen Parallelisierungsgrads innerhalb von Steuerungssystemen ist deren Nutzbarkeit zur Analyse des Gesamtsystems stark beschränkt.

Die Vorteile von Petri-Netzen sind die Darstellung von Parallelität, die formale Definition, die vielfältigen Analysemöglichkeiten und die hohe Bekanntheit. Abhängig der jeweiligen Nutzung existieren zahlreiche Ausprägungen. Offene Netze sind auf verteilte, asynchrone Softwaresysteme spezialisiert. Dadurch sind Schnittstellen zum technischen Prozess nicht berücksichtigt. NCES, SIPN und GRAFCET besitzen Schnittstellen zum technischen Prozess. NCES verfügen darüber hinaus über IT-Schnittstellen zur asynchronen und synchronen Kommunikation, welche über neu eingeführte Flussrelationen realisiert werden. Dies führt aber dazu, dass NCES Konventionen klassischer Petri-Netze nicht erfüllen, weshalb diese weniger verbreitet sind und von weniger Werkzeugen unterstützt werden.

Neben der Modellierungssicht auf das Verhalten eines Systems ist die Sicht auf die Abhängigkeiten innerhalb eines Systems für die Absicherung von Steuerungssystemen relevant, weshalb diese nachfolgend betrachtet werden.

Tabelle 1: Vergleich von Verhaltens-Modellierungsarten nach konzeptrelevanten Kriterien

Modellierungsart \ Kriterien	Darstellung von Parallelität	Spezialisierung-Modellierung von Logik	formalisierte Modellierungssprache	IT-Schnittstellen modellierbar	Schnittstellen zum technischen Prozess modellierbar	Zeitliches Verhalten modellierbar
Modelica	✓	◔	✓	✓	✓	✓
Zustands-automaten	✗	●	✓	✓	✓	✓
Klassische Petri-Netze	✓	●	✓	✗	✗	✓
Steuerungstechnisch interpretierte Petri-Netze	✓	●	✓	✗	✓	✓
Graphe Fonctionnel de Commande Etapes / Transitions	✓	●	✓	✗	✓	✓
Net Condition / Event Systems	✓	●	✓	✓	✓	✓
Offene Netze	✓	●	✓	✓	✗	✓
ARIS-Ereignisgesteuerte Prozessketten	✓	●	✗	✓	✗	✗
Business Process Model and Notation / Business Process Execution Language	✓	●	✓	✓	✗	✗

2.2.1.2 Darstellung von Abhängigkeiten verteilter Softwaresysteme

Aufgrund von informationstechnischen, physischen und sicherheitsrelevanten Beziehungen zwischen Komponenten existiert eine Vielzahl an Abhängigkeiten in verteilten Steuerungssystemen.

Zur Darstellung von Abhängigkeiten innerhalb verteilter Softwaresysteme sind in verschiedenen Domänen zahlreiche Modellierungsarten entstanden. Ausgehend von verschiedenen Rahmenbedingungen und bevorzugten Darstellungsarten unterscheiden sich die Modelle im Grad der Formalisierung, Ausdrucksfähigkeit und Komplexität.

Typische Domänen, bei welchen die Analyse von Abhängigkeiten innerhalb verteilter Softwaresysteme relevant ist, sind Geschäftsprozesse, die Softwareentwicklung sowie die Automatisierungstechnik. Ansätze dieser Domänen werden nachstehend vorgestellt und bewertet.

Prozesssicht

Bei verteilten Softwaresystemen treten prozessbedingt Abhängigkeiten auf. Somit können Abhängigkeiten über die in Kapitel 2.2.1.1 vorgestellten Verhaltensmodelle beschrieben und analysiert werden. So beschreibt [86] einen Ansatz, wie anhand von BPEL, formalisiert mit einer Petri-Netz-Semantik Abhängigkeiten innerhalb von Geschäftsprozessen analysiert werden können. Durch die detaillierte Darstellung und die starke Vernetzung der Modelle wegen der Vielzahl an Kommunikationsbeziehungen, besitzen diese Verhaltensmodelle entscheidende Nachteile bezüglich der Nachvollziehbarkeit von Abhängigkeiten.

Die Nachvollziehbarkeit der Abhängigkeiten lässt sich steigern, indem die komplexen Verhaltensmodelle vernachlässigt und nur Funktionsaufrufe zwischen Komponenten betrachtet werden [70]. Diese abstrakte Prozesssicht ist auf der linken Seite der Abbildung 6 anhand folgenden Beispiels verdeutlicht: Der Prozess A ruft nacheinander die Teilprozesse C, D, C, E auf. Der Prozess B ruft nacheinander die Teilprozesse E, C, D auf. Ein Vorteil der Prozesssicht liegt darin, dass die Reihenfolge der Prozessschritte dargestellt und somit die abhängigen nachfolgenden Prozesse erkannt werden können.

Diese Reihenfolge kann bei der Absicherung relevant sein. So müssen bei der Absicherung nur die Prozessschritte berücksichtigt werden, die dem geänderten Teilprozess nachfolgen. Dies ist aber nur möglich, wenn der Systemzustand bei Aufruf des geänderten Prozesses bekannt ist.

Ein typisches Beispiel im Bereich der Geschäftsprozesse ist die erweiterte ereignisgesteuerte Prozesskette (eEPK) des ARIS-Konzepts [70]. In [87] wurde die eEPK zur Modellierung von Service-Abhängigkeiten verwendet, da sie nach den Autoren nachvollziehbarer als BPEL ist. Ein typisches Beispiel für Prozessmodelle im Bereich der Automatisierungstechnik ist die formalisierte Prozessbeschreibung, bei welcher das Zusammenspiel zwischen Produkt, Prozess und Ressource während des Produktionsprozesses betrachtet wird [60]. Für die Software- und Systement-

wicklung eignet sich zur Prozessbeschreibung das in UML und SysML standardisierte Sequenzdiagramm [63], [68]. Das OASIS-Referenzmodel für serviceorientierte Architekturen definiert ebenfalls Prozess-Modelle, welche aber noch nicht ausreichend spezifiziert sind [54]. So ist beispielsweise die Orchestrierung verschiedener Services noch nicht modellierbar.

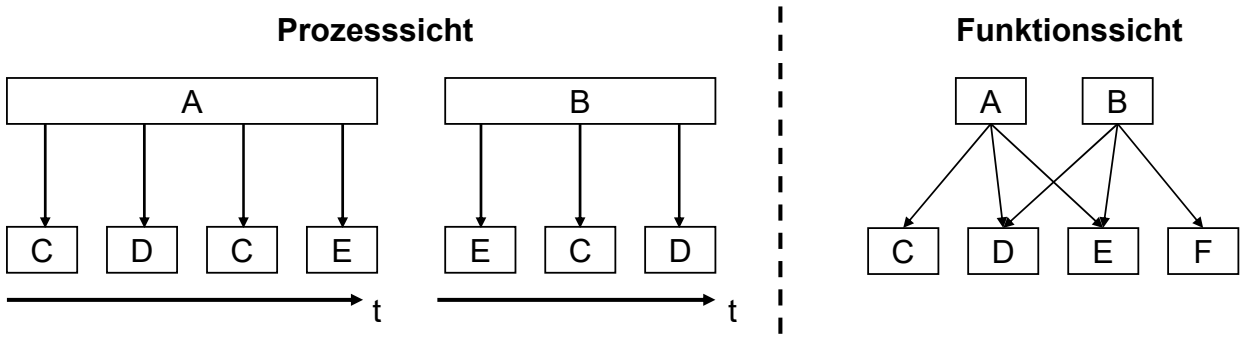


Abbildung 6: Darstellung der Abhängigkeiten zwischen Komponenten aus Prozess- und Funktionssicht

Funktionssicht

Alternativ sind Abhängigkeiten über eine hierarchische Funktionssicht darstellbar. Die Funktionssicht beschreibt nicht den zeitlichen Ablauf einer Funktionalität, sondern stellt dar, welche Teilfunktionalitäten für eine übergeordnete Funktionalität benötigt werden. Die in Abbildung 6 dargestellte Funktionssicht beschreibt das gleiche System wie die Prozesssicht. Daran wird deutlich, dass durch die Vernachlässigung zeitlicher Aspekte eine kompaktere Darstellung und dadurch eine höhere Nachvollziehbarkeit erreichbar ist. Besonders wenn viele Objekte wiederverwendet werden, ist die Funktionssicht deutlich schlanker als die Prozesssicht. So ist direkt erkennbar, dass von der Funktionalität D die Funktionalitäten A und B abhängen. Die Funktionssicht von hierarchisch verteilten Softwaresystemen ist über einen gerichteten, azyklischen Graphen darstellbar. Zyklen können nicht vorkommen, da dabei eine Komponente sich selbst aufrufen würde und diese Schachtelung in einer Endlosschleife resultieren würde [51].

Im Bereich der Geschäftsprozesse und betrieblichen Informationssysteme, existiert mit dem ARIS-Konzept eine mächtige Modellierungstechnik, welche in zahlreichen Werkzeugen verwendet wird [70]. Die Funktionssicht, welche in Abbildung 7 dargestellt ist, stellt eine von fünf Sichten des ARIS-Konzepts auf das zu beschreibende Informationssystem dar. Sie dient zur Darstellung der Beziehungen von Geschäftsprozessen innerhalb eines Informationssystems.

Im Bereich der Systementwicklung wurde in SysML das Blockdefinitionsdiagramm definiert, mit welchem Abhängigkeiten zwischen logischen oder physikalischen Komponenten über Abhängigkeitspfeile darstellbar sind. Wie in Abbildung 7 dargestellt werden Komponenten bei diesem Funktionsmodell durch Blöcke und Assoziationen über Pfeile modelliert. Das Blockdefinitionsdiagramm erlaubt die Darstellung der logischen oder physikalischen Dekomposition von Systemen und deren Spezifikation in jeder Entwicklungsphase. Entstehen Abhängigkeiten über einen

Funktionsaufruf werden die Abhängigkeitspfeile mittels <<call>> gekennzeichnet. Das Blockdefinitionsdiagramm ist das Pendant zum Klassendiagramm der UML und ermöglicht die Darstellung von Assoziationen, Aggregationen, Kompositionen und Generalisierungen [68].

In [88] wird beschrieben, wie UML-Klassendiagramme zur Unterstützung des Engineering-Prozesses von IEC-61499-konformen, verteilten Steuerungssystemen verwendet werden können. Mithilfe des UML-Klassendiagramms werden Software-Abhängigkeiten beschrieben. Aus dem Klassendiagramm lassen sich anschließend Funktionsblöcke nach IEC 61499 generieren.

Sicherheitsmodelle

Neben der Modellierung des Systems zur Abhängigkeitsanalyse sind im Bereich der Automatisierungstechnik zur Bestimmung der Zuverlässigkeit und Risiken von automatisierten Systemen die Fehlerbaumanalyse und die Ereignisablaufanalyse etabliert. Bei den Verfahren, welche in Abbildung 7 graphisch dargestellt sind, werden anstelle der Systemstruktur die Abhängigkeiten zwischen auftretenden Ereignissen beschrieben. Die Fehlerbaumanalyse ist eine Top-Down-Methode zur Beschreibung der Zusammenhänge zwischen Fehlern. Sind die Konsequenzen eines Fehlers bekannt, lässt sich somit ermitteln, welche Fehler zum Eintreten eines übergeordnete Ereignisses führen können (Deduktion) [89]. Die Ereignisablaufanalyse ist Bestandteil der Fehlermöglichkeits- und -einflussanalyse (FMEA). Sie dient bei der Sicherheitsbetrachtung zur Ermittlung aller Gefährdungen, die aus einem Fehler resultieren können. Durch ein induktives Vorgehen kann damit untersucht werden, wie sich ein Fehler auf weitere Ausfall-Ereignisse auswirkt [90]. Diese Verfahren erfordern zur Modellierung der Abhängigkeiten hohes Expertenwissen und müssen gesondert zur modellbasierten Entwicklung durchgeführt werden.

Alternative Darstellungsformen

Neben der graphischen Darstellung durch Modelle lassen sich Abhängigkeiten auch in Matrizen abbilden. Abhängigkeitsmatrizen bieten eine einfache Anpassbarkeit [91]. In [92] wird beschrieben, wie aus einem Abhängigkeitsgraph eine Abhängigkeitsmatrix generiert wird. Ein Ansatz zur Modellierung von Abhängigkeiten mithilfe einer Serviceabhängigkeitsmatrix wird in [91] beschrieben. Bei der Serviceabhängigkeitsmatrix werden Bedingungen definiert, unter welche ein Zustandsübergang stattfindet. Diese formalisierte Matrix muss während des Engineerings eines Moduls manuell erstellt werden. Nach Umwandlung in ein Petri-Netz lassen sich die modellierten Abhängigkeiten verifizieren.

Zur hoch formalen Darstellung von Abhängigkeiten kann auch die mächtige Prädikatenlogik verwendet werden. Diese, sich für komplexe Analysetechniken eignende, formale Sprache wird in [51] zur Beschreibung der Abhängigkeiten zwischen Produkten und Services verwendet. Die Sprache erfordert aufgrund der Vielzahl an logischen Ausdrücken eine hohe Fachkenntnis. Eine graphische Darstellung ist in [51] nicht gegeben.

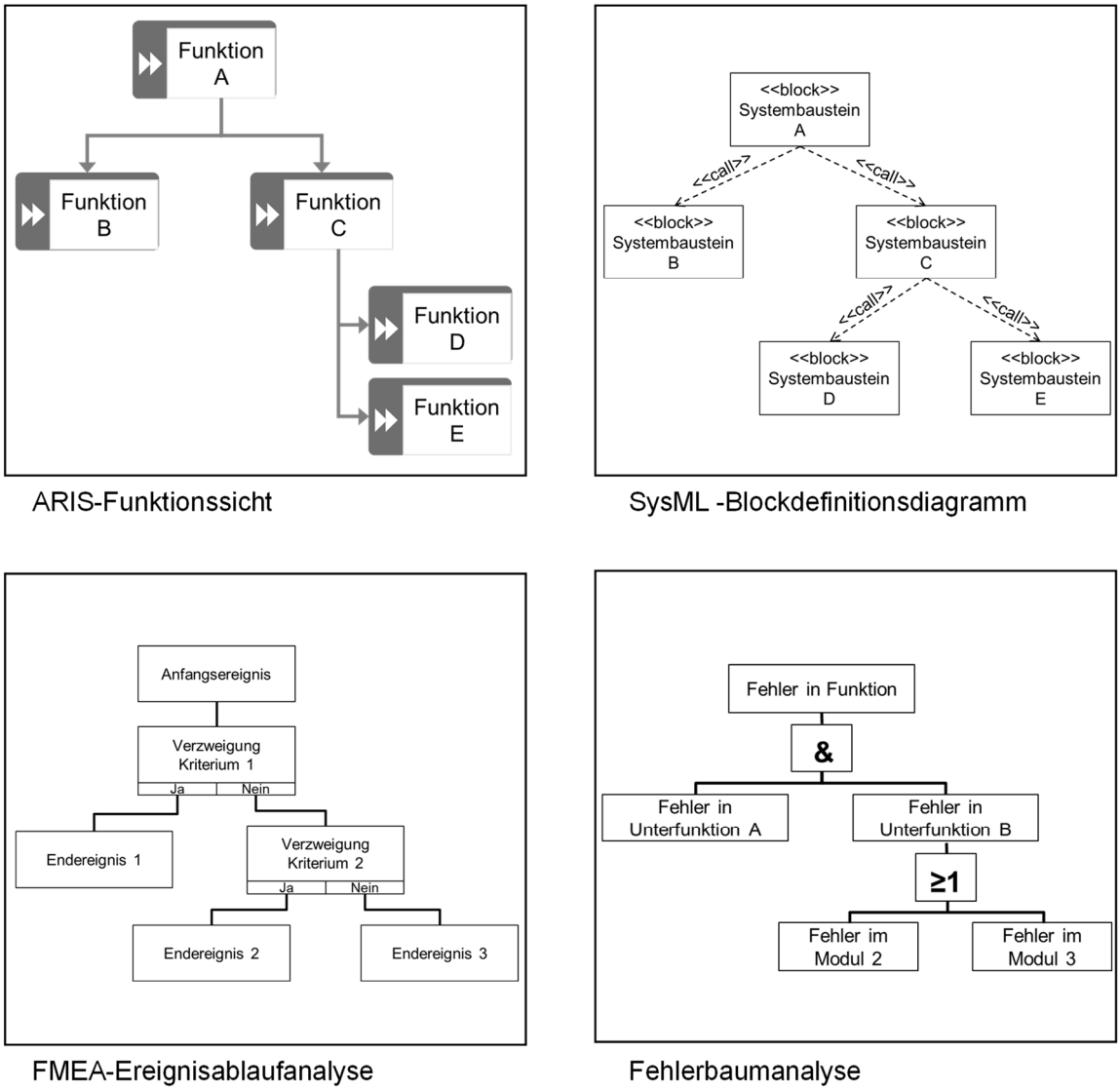


Abbildung 7: Diagramme zur Darstellung von Abhängigkeiten: ARIS-Funktionssicht [85]; SysML-Blockdefinitionsdiagramm[68]; FMEA-Ereignisablaufanalyse [90]; Fehlerbaumanalyse [89].

Zusammenfassende Bewertung der Diagramme zur Darstellung von Abhängigkeiten

Den vorgestellten Modellierungstechniken ist gemein, dass es sich um graphentheoretische Ansätze handelt. Dabei werden über gerichtete Kanten die Objekte in Beziehungen zueinander gesetzt. Dies erlaubt das maschinelle Prozessieren aller Modelle. Prozessmodelle können aufgrund der Berücksichtigung der Prozessreihenfolge Auswirkungen von Abhängigkeiten sehr gut eingrenzen, da die Fehlerauswirkung nur parallel ablaufende oder zeitlich nachfolgende Prozesse betreffen kann. Um diese Eingrenzung nutzen zu können, muss der Systemzustand bei Aufruf des geänderten Prozessschritts bekannt sein. Wesentlich übersichtlicher als Prozessmodelle sind Funktionsmodelle, die, aufgrund der Vernachlässigung zeitlicher Aspekte, Abhängigkeiten wesentlich aggregierter aufzeigen können. Sicherheitsmodelle sind zur Analyse von Gefahren und Zuverlässigkeit für viele Safety-kritische Anwendungen unverzichtbar, der Entwurf und die Analyse erfordern aber eine hohe Expertise und hohen personellen Aufwand.

Tabelle 2: Vergleich von Abhängigkeits-Modellierungsarten nach konzeptrelevanten Kriterien

Modellierungsart \ Kriterien	Standardisierung	Modellierungszweck	Formale Beschreibung	Nachvollziehbarkeit von Abhängigkeiten	Verbreitung	Domäne
UML – Sequenzdiagramm	ISO/IEC 19505	Zur Darstellung der Interaktion zwischen Komponenten. Der Austausch von Nachrichten zwischen Objekten wird mittels Lebenslinien dargestellt.	✓	◔	●	Software-Entwicklung & System-entwurf
Formalisierte Prozess-beschreibung [57]	VDI/ VDE 3682	Vorgehen zur interdisziplinären Beschreibung von Produkt, Prozess und Ressource über den Lebenszyklus.	✓	◑	◔	Automati-sierungs-technik
UML - Klassendiagramm	ISO/IEC 19505	Zur Strukturierung von Software bei der Entwicklung. Darstellung der Beziehungen zwischen Klassen und der Schnittstellen.	✓	●	●	Software-entwicklung
SysML – Blockdefinitions-diagramm	ISO/IEC 19514: 2017	Zur Strukturierung komplexer Systeme. Darstellung der Abhängigkeiten und Verbindungen zwischen Komponenten.	✓	●	◕	System-entwicklung
ARIS-Funktions-baum (Architektur integrierter Informations-systeme) [89]	-	Statische Zerlegung übergeordneter Funktionen.	✓	●	◑	Betriebliche Informations-systeme für Geschäfts-prozesse
FMEA – Ereignisablauf-analyse	DIN 25419	Sicherheitsbetrachtung zur Ermittlung aller Gefährdungen, die aus einem Fehler resultieren können. Induktives Verfahren zur Darstellung der Abhängigkeiten zwischen Ausfall-Ereignissen.	✓	●	◑	Zuverlässig-keitstechnik
Fehlerbaum-analyse (FTA)	DIN 25424-1	Analyse von Fehlerursachen, die zu unerwünschten Ereignissen führen können. Deduktives Verfahren zur Darstellung der Abhängigkeiten zwischen Ausfall-Ereignissen.	✓	●	◑	Zuverlässig-keitstechnik

2.3 Testen der Software von Steuerungssystemen

„Mit dem zunehmenden Eindringen von Computern in viele Anwendungsbereiche gewinnt die Sicherstellung der korrekten, zuverlässigen Funktion der verwendeten Software an Bedeutung“ [93]. Zur Sicherstellung einer ausreichenden Software-Qualität kommt dem Testen eine wesentliche Rolle zu. Es dient der Vermeidung und dem Finden von Softwarefehlern. Dabei beinhaltet Testen weitaus mehr als das Ausführen von Testfällen und wird vom International Software Testing Qualifications Board (ISTQB) wie folgt definiert [4]:

Testen: „Der Prozess, der aus allen Aktivitäten des Lebenszyklus besteht (sowohl statisch als auch dynamisch), die sich mit der Planung, Vorbereitung und Bewertung eines Softwareprodukts und dazugehöriger Arbeitsergebnisse befassen. Ziel des Prozesses ist [zum einen] sicherzustellen, dass diese allen festgelegten Anforderungen genügen, dass sie ihren Zweck erfüllen, und [zum anderen] etwaige Fehlerzustände zu finden.“

In diesem Kapitel wird ein Überblick über das Softwaretesten sowie Forschungsbestrebungen im Bereich des modellbasierten Tests und der formalen Verifikation in der Domäne Automatisierungstechnik gegeben. Dazu wird zunächst erläutert, was ein Softwarefehler ist. Darauf aufbauend werden grundlegende Software-Testverfahren vorgestellt, welche dazu dienen, Softwarefehler zu identifizieren. Im Folgenden wird auf einige Forschungsbestrebungen eingegangen, welche die Testverfahren verwenden, um strukturierte und effiziente Testmethoden zu konzipieren. Dazu werden zuerst die Forschungsbestrebungen im Bereich modellbasiertes Tests und anschließend im Bereich der Anwendung von formalen Verifikationsmethoden vorgestellt.

2.3.1 Softwarefehler

Testen dient häufig dem Auffinden von Fehlern. Dabei wird der Begriff „Fehler“ abhängig von der Domäne verschieden definiert. Recht allgemein definiert die ISO 9000 den Fehler als

„Nichterfüllung einer Anforderung“ [94].

Im Bereich des Softwaretests differenziert das ISTQB zwischen dem Fehlerzustand und einer möglicherweise daraus resultierenden sichtbaren Fehlerwirkung [4]:

Fehlerzustand: „Defekt (innerer Fehlerzustand) in einer Komponente oder einem System, der eine geforderte Funktion des Produkts beeinträchtigen kann, z.B. inkorrekte Anweisung oder Datendefinition. Ein Fehlerzustand, der zur Laufzeit angetroffen wird, kann eine Fehlerwirkung einer Komponente oder [eines] Systems verursachen.“

Fehlerwirkung: „Ein Ereignis, in welchem eine Komponente oder ein System eine geforderte Funktion nicht im spezifizierten Rahmen ausführt.“ (nach ISO 24765 [95])

2.3.2 Grundlegende Software-Testverfahren

Der Softwaretest ist Teil der analytischen Qualitätssicherung, welche für ein spezifisches Software-Produkt eine Fehlerfindungsstrategie definiert. Wie in Abbildung 8 dargestellt, wird zwischen statischem Test und dynamischem Test unterschieden. Während beim dynamischen Test das Software-Produkt ausgeführt wird, wird beim statischen Test das Software-Produkt ohne Ausführung begutachtet. Statische Tests werden meist vor dynamischen Tests ausgeführt und dienen dazu, Fehlerzustände zu identifizieren und Fehlerzuständen vorzubeugen. Zum statischen Test gehören Reviews, welche manuell ausgeführt werden und unterschiedliche Ausprägungen besitzen können – von einem informellen Review bis zu einer Inspektion. Neben Reviews gehören statische Analyseverfahren zum statischen Testen. Bei der statischen Analyse wird der Programmcode oder dessen Modell werkzeugunterstützt nach definierten Kriterien untersucht. So kann mittels formaler Beweismethoden, wie der formalen Verifikation, die Konformität einer Software gemäß ihrer formalisierten Anforderungen mathematisch nachgewiesen werden. Weiter werden statische Analysen von Compilern zur Datenflussanalyse, Kontrollflussanalyse und Berechnung von Komplexitätsmetriken verwendet [10].

Im Gegensatz zum statischen Testen zielt das dynamische Testen darauf ab, statt den Fehlerzuständen die Fehlerwirkungen zu identifizieren. Dabei werden Testfälle auf einem ablauffähigen Software-Produkt ausgeführt. Die Erstellung von Testfällen basiert beim White-Box-Test auf der inneren Struktur eines Software-Produkts. Im Gegensatz dazu erfordern Black-Box-Tests keine Kenntnis der inneren Struktur eines Software-Produkts. Testfälle werden dabei aus Anforderungen an das Produkt und aus Anwendungsfällen abgeleitet [10].

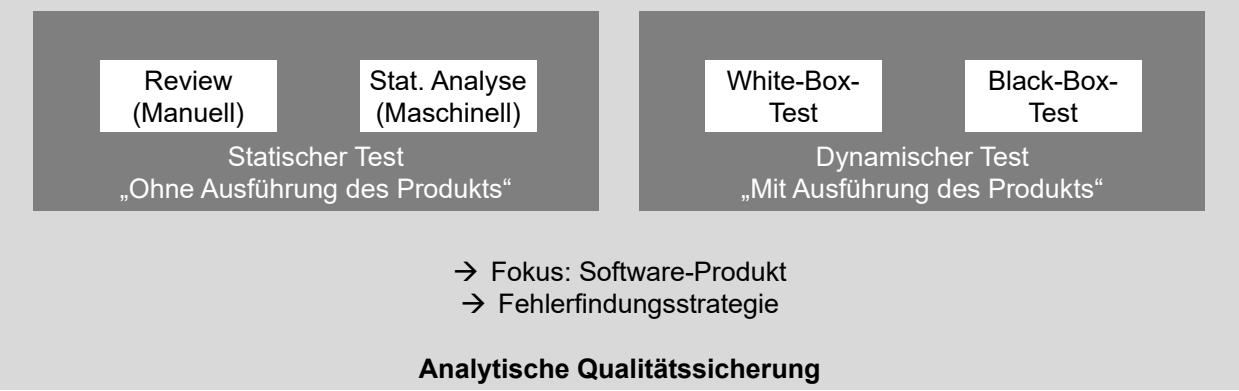


Abbildung 8: Überblick über grundlegende Testverfahren nach [96]

Bei den Anforderungen an das Software-Produkt wird zwischen funktionalen und nicht-funktionalen unterschieden. Nach ISO 24765 spezifizieren funktionale Anforderungen, welche Funktion eine Komponente oder ein System verwirklichen muss. Demnach beschreiben nicht-funktionale Anforderungen, wie eine Komponente oder ein System ihre beziehungsweise seine beabsichtigte Leistung erbringen soll [95].

2.3.2.1 Formale Verifikation

Aufgrund der automatisierten Ausführbarkeit und der Eignung für sicherheitskritische Anwendungen wird nachfolgend die Methode der formalen Verifikation beschrieben und es werden ihre Vor- und Nachteile erörtert. Statt, wie beim dynamischen Testen, mit Testfällen eine Software stichprobenartig zu testen, werden anhand des Programmcodes oder dessen Modells die Mechanismen, welche beim Programmablauf wirken, mittels mathematischer Berechnungen analysiert. Mit diesem formalen Korrektheitsbeweis kann die Einhaltung von Systemeigenschaften garantiert werden. Dazu müssen die Anforderungen, gegen die geprüft werden soll, in formaler Sprache, zum Beispiel in temporaler Logik, beschrieben sein [93].

Da Automaten ein etabliertes Beschreibungsmittel sind, werden insbesondere in der Automatisierungstechnik automatenbasierte Verifikationstechniken eingesetzt [93]. Die Anwendung von automatenbasierten Techniken, wie dem Symbolic Model Checking, erfordert eine streng formale Darstellung der Modelle und Anforderungen. Einige formale Modellierungsarten sind in Kapitel 2.2.1.1 beschrieben. Das Symbolic Model Checking beschreibt eine vollautomatisierte Verifikation einer Systembeschreibung gegen seine formal spezifizierten Anforderungen. Dies ermöglicht die Analyse der Systembeschreibungen nach folgenden Kriterien [93]:

- Sequenzen: Existiert eine gewünschte Zustandsübergangssequenz?
- Lebendigkeit: Ist jede Transition in endlicher Zeit wieder schaltbar?
- Reversibilität: Kann der Ausgangszustand wieder eingenommen werden?
- Partielle und totale Verklemmung (Livelock / Deadlock): Ist es möglich, dass das System in einen Zustand kommt, ab welchem in endlicher Zeit nur Teilfunktionalitäten oder keine Funktionalität mehr ausführbar ist?
- Erreichbarkeit: Kann ein gewünschter oder gefährlicher Zustand erreicht werden?

Das Symbolic Model Checking überprüft den erreichbaren Zustandsraum. Durch Finden eines Gegenbeispiels kann die Nichterfüllung einer Anforderung festgestellt werden. Zur Steigerung der Effizienz des Verifikationsprozesses existieren leistungsfähige Algorithmen, welche möglichst geschickt den Zustandsraum aufspannen und durchsuchen [93].

Anforderungen, welchen ein System genügen muss, lassen sich mithilfe von Temporallogiken, wie CTL (Computation Tree Logic) und LTL (Linear Temporal Logic), formal ausdrücken. CTL erlaubt die Beschreibung des Soll-Verhaltens eines Systems ausgehend von einem Initialzustand durch Definition von Zustandssequenzen (Pfaden) [97]. Das Ergebnis der Überprüfung einer in Temporallogik beschriebenen Anforderung, beispielsweise über Model Checking, gibt Auskunft darüber, ob die spezifizierte Eigenschaft für den gesamten Zustandsraum gültig ist [93].

Werkzeuge zur Durchführung des Model Checking, sogenannte Model Checker, stammen hauptsächlich aus dem universitären Umfeld. Verbreitete Model Checker sind: SPIN, NuSMV, ITS-Tool, SESA, KRONOS, UPPAAL und HyTech. Alle aufgelisteten Model Checker unterstützen die Verifikation von Zustandsautomaten und Petri-Netzen gegen Anforderungen, welche über Temporallogiken, wie CTL und LTL, spezifiziert sind.

Aufgrund einiger Nachteile, welche in Abbildung 9 dargestellt sind, finden formale Verifikationsverfahren in der Praxis häufig keine Anwendung. Oftmals sind die Rahmenbedingungen zur Verifikation, wie ein geeignetes Modell oder formalisierte Anforderungen, nicht erfüllt. Darüber hinaus führen große Verhaltensmodelle mit vielen alternativen und parallelen Abläufen zur sogenannten Zustandsraumexplosion, was zu rechenintensiven Verifikationsprozessen führt [98]. Des Weiteren kann, da bei der formalen Verifikation die Software-Produkte nicht ausgeführt werden, keine Aussage über deren Laufzeitverhalten getroffen werden.

Demgegenüber stehen zahlreiche Vorteile. Ein großer Vorteil der formalen Verifikation ist, dass es sich um eine mathematische exakte Beweisführung handelt und somit keinen stichprobenartigen Charakter wie das dynamische Testen aufweist. Aufgrund langjähriger Forschung im Bereich der Informatik stehen zudem zahlreiche ausgereifte Verifikations-Werkzeuge zur Verfügung. Anhand dieser Werkzeuge ist eine vollautomatisierte Berechnung des Verifikationsergebnisses möglich. So kann ohne manuelle Tätigkeiten die Konformität der Logik eines Software-Produkts zu dessen formalisierten Anforderungen bewiesen werden [98]. Darüber hinaus ist, da man das Modell und nicht die Anlage testet, eine Überprüfung vor Inbetriebnahme möglich. Bei dieser Vorabprüfung besteht kein Beschädigungsrisiko für das Automatisierungssystem.

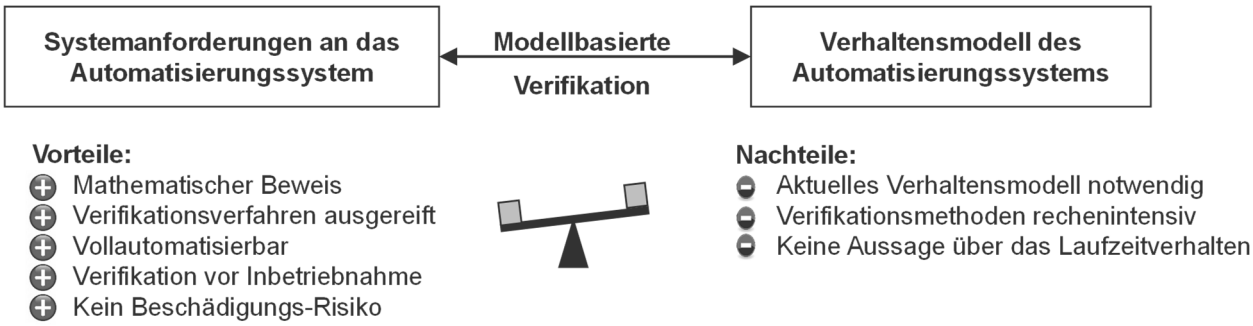


Abbildung 9: Vor- und Nachteile modellbasierter Verifikationsmethoden

Folglich wird die formale Verifikation häufig verwendet, wenn eine hohe Software-Qualität gefordert ist. Dabei handelt es sich oft um sicherheitskritische Anwendungen oder Software-Produkte, bei welchen Softwarefehler im Betrieb hohe Kosten im Falle eines Systemausfalls oder bei Rückrufaktionen verursachen können. Formale Methoden können also eine sinnvolle Ergänzung zu dynamischen Tests darstellen, werden aber noch sehr selten eingesetzt.

2.3.3 Forschungsbestrebungen modellbasierter Test

Das modellbasierte Testen beschreibt die Nutzung von Modellen zur Darstellung von Testartefakten und zur Automatisierung des Testprozesses [62]. Als Testprozess wird dabei meist der fundamentale Testprozess des ISTQB adressiert, welcher in Abbildung 10 mitsamt den in den Phasen entstehenden Testartefakten dargestellt ist.

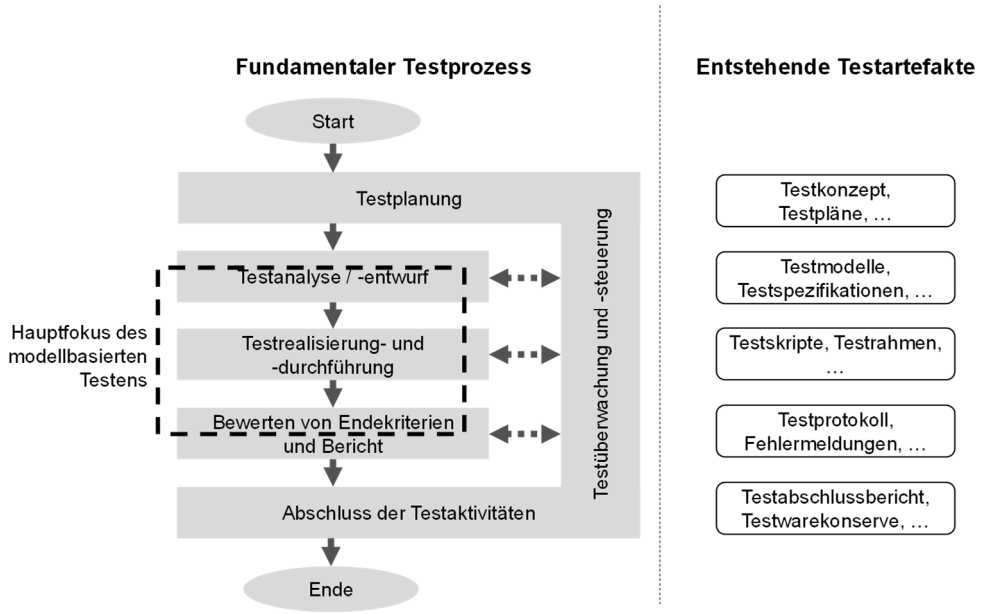


Abbildung 10: Fundamentaler Testprozess und Testartefakte, welche während des Testprozesses anfallen nach [10]

Die Hauptaktivitäten im Bereich des modellbasierten Testens sind in Abbildung 11 dargestellt. Bei der Formalisierung werden nicht formale Anforderungen des Lasten- oder Pflichtenhefts in formalen Spezifikationsmodellen abgebildet. Aus diesen abstrakten Modellen werden Testfälle generiert, welche auf einem Testsystem automatisiert ausgeführt werden können [99]. Somit wird modellbasiertes Testen hauptsächlich als Teil des dynamischen Testens verstanden.

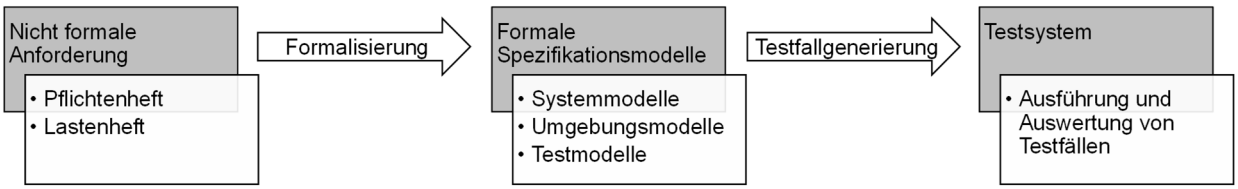


Abbildung 11: modellbasierter Testprozess nach [99]

Oftmals werden zur Modellierung beim modellbasierten Testen die UML oder Petri-Netze genutzt. Da die UML 2.0 zur Spezifikation der Testartefakte zum Beispiel der Testfälle, nicht ausreicht, wurde das Testprofil U2TP (UML Testing Profile) standardisiert, welches UML 2.0 erweitert [100].

Im Feld der Automatisierungstechnik gibt es zahlreiche Forschungsvorhaben, welche sich in den fundamentalen Testprozess einordnen lassen.

Die Forschungsbestrebungen der Phase Testanalyse / -entwurf beziehen sich hauptsächlich auf die Formalisierung von Anforderungen und Generierung von Testfällen. Zur Testfallgenerierung wird ein Spezifikationsmodell, welches die Anforderungen an ein System beinhaltet, benötigt. Es beschreibt das Soll-Verhalten der zu testenden Software sowie deren Umgebung [101]. Aufgrund der hohen Komplexität bei der Erstellung eines Spezifikationsmodells existieren zahlreiche Ansätze, die Modellerstellung zu automatisieren.

Magnus et al. beschreiben in [99], [102] ein Konzept, um Anforderungen, welche einzeln als Sequenzdiagramm beschrieben werden, in ein Spezifikationsmodell, zum Beispiel einen Zustandsautomaten, zu überführen. Damit aus Spezifikationsmodellen Testfälle generiert werden können, muss es das Verhalten des Systems ausreichend widerspiegeln. Dazu werden die Sequenzdiagramme aller Anforderungen in einem Zustandsautomaten synthetisiert. Dabei wird der Zustandsautomat iterativ erweitert. Um aus dem Zustandsautomat eine, für vorgegebene Features geeignete, Testsuite zu generieren, wurde dieser Ansatz um eine Testfallpriorisierung erweitert [103].

Krause konzipierte zur Darstellung eines Spezifikationsmodells das Beschreibungsmittel SPENAT (Sicheres Petri-Netz mit Attributen), welches auf Petri-Netzen basiert. Dabei wird ausgenutzt, dass zur Analyse von Petri-Netzen zahlreiche Verfahren existieren. Dies erlaubt die gezielte Generierung von Testfällen, um eine möglichst hohe Testabdeckung zu erzielen. Zur Testrealisierung lassen sich diese Testfälle anschließend automatisiert in ausführbare Testskripts, z.B. der Programmiersprache TTCN-3 (Testing and Test Control Notation), umwandeln [104]. Neben SPENAT gibt es von Provost weitere Arbeiten, bei welchen GRAFCET-Modelle, die ebenso auf Petri-Netzen basieren, zur Testfallgenerierung verwendet werden [105].

In den Arbeiten von Hametner et al. [106] und Hussain et al. [107] werden weitere Verfahren vorgestellt, wie aus UML-Diagrammen Testfälle für IEC61499-Anwendungen generiert werden können.

Neben dem Prüfen des Soll-Verhaltens eines Systems können mit Testfällen auch die Fehlerbehandlungsroutinen eines Steuerungssystems überprüft werden. Rösch präsentiert in [108] ein Verfahren, bei welchem aus Sequenzdiagrammen, die das Soll-Verhalten eines Steuerungssystems beschreiben, Testfälle abgeleitet werden, die anhand gezielter Fehlerinjektion Fehlerbehandlungsroutinen eines Steuerungssystems testen.

Zur Testrealisierung existieren zahlreiche Generatoren, welche aus den modellierten Testfällen ausführbare Testskripte generieren. Diese erzeugen meist ausführbare Testskripte der Sprache TTCN-3 aus U2TP, GRAFCET oder SPENAT-Modellen. Kormann et al. beschreiben in [109] wie aus in UML spezifizierten Testfällen automatisch SPS-Code generiert und zur Testdurchführung auf eine SPS übertragen werden kann. Neben der Ausführung an realen Anlagen können die Testskripte zum Testen einer virtuellen Anlage im Rahmen der virtuellen Inbetriebnahme verwendet werden [110].

Bewertung relevanter Forschungsbestrebungen im Bereich des modellbasierten Tests

Das modellbasierte Testen ermöglicht eine Automatisierung des Testprozesses von der Formalisierung der Anforderungen bis zur Testdurchführung. Die vorgestellten Ansätze nutzen maschineninterpretierbare Modelle, um aufwendige und fehleranfällige Prozessschritte zu automatisieren. Die Strukturierung und Automatisierung des Testprozesses ermöglicht eine hohe Effizienz und einen umfassenden Test.

Zur Formalisierung von Anforderungen, Generierung und Durchführung von Testfällen existieren bereits ausgereifte Konzepte. Bei der Testdurchführung wird ein Testsystem verwendet, welches automatisiert das zu testende System stimuliert und überwacht. Je nach Ansatz ist das zu testende System real oder virtuell.

Nachteile bei der Testdurchführung an einer realen Anlage sind die schwierige Reproduzierbarkeit von Testergebnissen, die Verfügbarkeit und das Beschädigungsrisiko. Virtuelle Testobjekte, wie bei der virtuellen Inbetriebnahme, erfordern ein ausreichend genaues Modell der realen Anlage. Diese Modelle des Testobjekts und deren Umgebung werden bei den vorgestellten Ansätzen als gegeben angenommen. Aufgrund des stichprobenhaften Charakters des dynamischen Testens ist zudem eine vollständige Testabdeckung nie erreichbar. Daher werden klassische, dynamische Tests bei sicherheitskritischen Anwendungen durch die formale Verifikation ergänzt.

2.3.4 Forschungsbestrebungen formale Verifikation

Die formale und somit exakte Verifikation von Steuerungssoftware wird in der Forschung seit Jahrzehnten diskutiert. Wurde anfangs hauptsächlich die Verifikation von SPS-Programmcode betrachtet, verschob sich im letzten Jahrzehnt der Fokus der Forschung auf verteilte, änderbare Steuerungssysteme. Bei Forschungsbestrebungen im Bereich der Automatisierungstechnik stehen weniger die formalen Verifikationsalgorithmen als die verwendeten Modellierungstechniken im Vordergrund. Dabei wird meist das Modell der Steuerung mit dem Modell des zu steuernden technischen Prozesses als geschlossener Kreislauf (Closed-Loop) verifiziert. Im Folgenden werden für die Arbeit relevante Konzepte zur Anwendung formaler Verifikationsmethoden vorgestellt.

Die Verifikation von SPS-Programmen nach IEC 61131-3 ist schon ausgiebig erforscht. Für einen Überblick über gängige Methoden zur Verifikation von SPS-Programmen sei auf das Survey von Frey et al. [111] verwiesen.

Vyatkin et al. [112] adressieren die Herausforderung, dass oftmals Modelle des Steuerungsprogramms nicht in geeigneter Form vorliegen und dies ein Hindernis bei der Anwendung von Verifikationsmethoden darstellt. Dies wird explizit für SPS-Code betrachtet, welcher als Kontaktplan vorliegt. Dazu wird ein Generator vorgestellt, welcher automatisiert aus dem Kontaktplan ein zur Verifikation geeignetes Net Condition/Event System (NCES) generiert. Um eine Closed-Loop-Verifikation zu ermöglichen, wird der technische Prozess diskretisiert und ebenfalls als NCES abgebildet. Hierbei ermöglichen definierte Schnittstellen die Komposition der Modelle der Steuerung und des technischen Prozesses.

Darüber hinaus wurde von Preuße et al. ein wiederverwendbares Framework entwickelt, um Risiken bei der Inbetriebnahme durch formale Methoden zu reduzieren [113]. Dazu wird ebenfalls eine Closed-Loop-Verifikation durchgeführt, bei welcher die Steuerung und der technische Prozess als NCES modelliert sind. Zur Modellierung des technischen Prozesses wird dabei ein komponentenbasierter Ansatz gewählt. Zur Modellierung der Verbindungen zwischen den Modulen sind weitere Informationen notwendig, weshalb die Komposition nur semi-automatisiert durchgeführt werden kann.

Eine Übersicht über Verifikationsverfahren für Agentensysteme geben Wassermann et al. in [114]. Darin werden unter anderem Ansätze beschrieben, welche die Konvertierung von agentenspezifischen Modellierungssprachen in verbreitete Modellierungssprachen wie farbige Petri-Netze betrachten oder bei denen Petri-Netze zur Beschreibung von Agenten erweitert werden. Ferner werden Temporallogiken, wie CTL, zur Beschreibung für Agenteneigenschaften erweitert.

Als Grundlage zur Verifikation verteilter Systeme wird oftmals die in IEC 61499 beschriebene Architektur verwendet. Diese basiert auf dem Konzept der Funktionsblöcke, welche eine komponentenbasierte Modellierung des Steuerungssystems ermöglicht [115]. Forschungsbestrebungen zur Verifikation von IEC-61499-basierten Systemen durch formale Verifikation werden von Blech et al. in [115] dargestellt und verglichen. Dies beinhaltet zahlreiche Ansätze zur automatischen Überprüfung von Systemeigenschaften nach Hardware-Rekonfigurationen, welche teilweise Softwareanpassungen erfordern [116]–[118]. Sie basieren auf einer Closed-Loop-Modellierung, welche neben dem Verhalten des Steuerungssystems auch das physische Verhalten des technischen Systems miteinschließt. Zur Verifikation des Verhaltens der in IEC 61499 definierten Funktionsblöcke wird deren Verhalten oftmals als NCES modelliert (Ivanova-Vasileva et al. [119], Hanisch et al. [120]). Da Verhaltensmodelle häufig als Automaten vorliegen, wurden zudem Modellgeneratoren implementiert, welche Automaten automatisiert in NCES übersetzen.

Die formale Verifikation von rekonfigurierbaren Systemen wird in weiteren Ansätzen explizit berücksichtigt. In [97] betrachtet Khalgui die Modellierung und Verifikation verteilter, rekonfigurierbarer Steuerungssysteme. Dafür wird vorausgesetzt, dass die möglichen Konfigurationen im Voraus bekannt und modelliert sind. Zur Abbildung des Einflusses der Rekonfigurationen auf das Systemverhalten wird eine aus mehreren Ebenen bestehende Architektur entworfen. Die Logik in den Ebenen, welche miteinander verknüpft sind, wird einheitlich als NCES modelliert. Das Modell der obersten Ebene stellt dar, welche Konfiguration aktuell aktiv ist. Durch die Verknüpfungen der Modelle werden Teilbereiche des Verhaltensmodells des Steuerungssystems aktiviert, wodurch das Verhaltensmodell das Systemverhalten bei der jeweiligen Konfiguration widerspiegelt. Das Verhalten lässt sich anschließend mittels eines Model Checkers verifizieren. In [121] beschreibt Zhang die Anwendung von NCES zur Modellierung und Verifikation rekonfigurierbarer ereignisdiskreter Steuerungssysteme. Dazu werden die Netze zur R-TNCES (Reconfigurable Timed Net Condition/Event System) erweitert, mit welchen sich Rekonfigurationsszenarien modellieren und recheneffizienter darstellen lassen. In [122] wird eine zusätzliche Erweiterung von R-TNCES zu „extended R-TNCES“ und deren Nutzung zur Verifikation beschrieben. Die Erweiterung ermöglicht die Darstellung zeitgleicher Rekonfigurationen verschiedener Teilsysteme. Diesen Verfahren ist gemein, dass mögliche Konfigurationen des Systems im Voraus bekannt und modelliert sein müssen.

Eine Methodik zur formalen Entwicklung verteilter, reaktiver Systeme wird in [69] von Broy et al. vorgestellt. Die streng formale Darstellung der Entwicklungsergebnisse erlaubt deren Konsistenzprüfung in verschiedenen Entwicklungsstufen und auf verschiedenen Abstraktionsebenen anhand formaler Verifikationsmethoden. Die Modelle der Komponenten des verteilten Systems kooperieren dabei über syntaktisch und semantisch definierte Schnittstellen. Sie adressiert hauptsächlich die Entwicklungsphase. Rahmenbedingungen der Automatisierungstechnik und der Betriebsphase werden nicht explizit berücksichtigt. Der Vorteil der Verifizierbarkeit der Arbeitsergebnisse sämtlicher Entwicklungsphasen wird durch eine hohe Komplexität der Modellierung erkauft [70].

Im Rahmen des DFG Schwerpunktprogramms 1593 wird unter anderem die Verifikation von sich ändernder Software aus verschiedenen Blickwinkeln betrachtet. So beschreiben Legat et al. [123] ein Verfahren zur Verifikation von Schnittstellen-Verhaltensmodellen komponentenbasierter Systeme nach Änderungen unter Berücksichtigung der Hardware, Software und Elektrik. Die verwendeten Verhaltensmodelle beinhalten Material-, Energie-, physikalische sowie Datenschnittstellen und sind somit deutlich mächtiger als es zur Verifikation von Steuerungsprogrammen notwendig ist. In [17] wird beschrieben, wie Änderungen an Systemen zur Laufzeit erkannt und eine stetige Verifikation durchgeführt werden kann. Dazu werden Prozessdaten analysiert, bei welchen bekannt ist, welche Systemanforderungen sie adressieren. Um dies zu ermöglichen, werden Struktur- und Verhaltensmodelle aus Prozessdaten abgeleitet, welche über ein Informationsmodell semantisch beschrieben sind. Darüber hinaus wird im Rahmen des Schwerpunktprogramms ein

Konzept vorgestellt, welches die Effizienz bei Verifikation von variantenreicher Software steigert. Dazu werden die Pfade, die sich in ihren Varianten unterscheiden, identifiziert und über ein inkrementelles Model Checking geprüft [124]. Ladiges et al. untersuchen die Verifikation während der Betriebsphase. Das beschriebene Verifikationsverfahren basiert auf der Auswertung aufgezeichneter Prozessdaten [125]. Das Schwerpunktprogramm 1593 adressiert mit unterschiedlichen Ansätzen die Evolution von Software über den Softwarelebenszyklus. So existieren auch Ansätze, welche Änderungen während der Betriebsphase fokussieren. Diese basieren auf der Auswertung von Prozessdaten. Dies verhindert eine Vorabprüfung, bevor ein Automatisierungssystem nach vorgenommenen Änderungen erneut in Betrieb genommen wird.

Tabelle 3: Bewertung der Ansätze zur Verifikation nach in der Zielsetzung definierten Nebenbedingungen

Kriterien / Ansätze	Eignung für wandelbare Produktionsnetzwerke	Integration in ein Steuerungssystem vorgesehen	Fokus Betriebsphase	hoher Automatisierungsgrad bei der Anwendung
Vyatkin [112]	✓	✗	✗	✓
Preuße [113]	✓	✗	✗	✗
Hanisch [120]	✓	✗	✗	✓
Khalgui [97]	✓	✗	✗	✗
Broy [69]	✓	✗	✗	✗
Ladiges [125]	✓	✗	✓	✓

Bewertung relevanter Forschungsbestrebungen im Bereich der formalen Verifikation

Die vorgestellten Ansätze wurden nach den in Kapitel 1.3 definierten Nebenbedingungen zur Beantwortung der Forschungsfrage und zur Erreichung der definierten Zielsetzung analysiert. Die Bewertung der Ansätze ist in Tabelle 3 zusammengefasst. Die meisten Ansätze der formalen Verifikation basieren auf der Beschreibung von Modellierungstechniken für verschiedene Systeme und stellen innovative Ansätze dar, um wandelbare Steuerungssysteme abzubilden.

Aus der Bewertung wird aber auch deutlich, dass die meisten Ansätze sich nicht auf die Betriebsphase konzentrieren. Entweder wird die formale Verifikation während des Engineerings oder sie wird nur rein methodisch betrachtet. So ist die Verknüpfung zu einem konkreten Anwendungsfall meist nicht gegeben. Es existiert kein Ansatz, welcher explizit den Anwendungsfall betrachtet,

wie Anlagenbetreiber bei der Anwendung formaler Verifikationsmethoden unterstützt werden können.

Zur Anwendung der Verfahren ist oftmals ein hoher manueller Aufwand notwendig. Dies liegt daran, dass die notwendigen Modelle erstellt und Schnittstellenbeschreibungen definiert werden müssen. So wird oftmals kein Vorschlag gegeben, wie die notwendigen Modelle automatisiert generiert werden können. Dies erlaubt nur eine semi-automatisierte Anwendung der Verfahren. Aufgrund der rein methodischen Darstellung wird auch selten eine Aussage über die Nachvollziehbarkeit von Fehlerursachen aus den Verifikationsergebnissen gegeben.

Viele Ansätze orientieren sich an der Verifikation von Speicherprogrammierbaren Steuerungen (SPS). So werden oftmals Funktionsbausteine nach IEC 61131-3 oder IEC 61499 betrachtet, welche modelliert und verifiziert werden.

Da die Systeme meist nicht den aktuellen Zustand eines Steuerungssystems identifizieren, besteht kein Bedarf zur Integration des Verifikationssystems in ein Steuerungssystem.

2.4 Zusammenfassende Bewertung der Verfahren und Folgerung

Die vorgestellten Konzepte stellen allesamt interessante Lösungsansätze dazu dar, wie eine Verbesserung des Absicherungsprozesses erreicht werden kann. So konnten Fortschritte zur Automatisierung, Strukturierung und Systematisierung des Testprozesses erreicht werden. Anhand der Bewertung der Ansätze im Bereich des modellbasierten Testens und der formalen Verifikation wird aber auch deutlich, dass sich die meisten Ansätze auf die Engineeringphase beziehen oder rein methodisch angelegt sind.

So fehlt es an Ansätzen, die explizit die Absicherung von Änderungen in der Betriebsphase adressieren. Insbesondere da Anlagenbetreiber meist nicht über große Softwaretest-Erfahrung verfügen, bedarf es Konzepte, die Anlagenbetreiber durch Automatisierung bei der Absicherung von Automatisierungssystemen nach Änderungen an der Steuerungssoftware unterstützen.

Formale Methoden, wie das Model Checking, sind zur Absicherung einiger sicherheitskritischer Anwendungen etabliert. So existieren schon zahlreiche Werkzeuge, welche vollautomatisiert eine Verifikation berechnen. Aufgrund fehlender oder veralteter Verhaltensmodelle werden formale Verifikationsverfahren aber meist nicht angewendet. Zielsetzung dieser Arbeit ist es, diese Technologie für Anlagenbetreiber nutzbar zu machen.

Die hohe Komplexität bei Anwendung der formalen Verifikation liegt in der Bereitstellung der für die Verifikation notwendigen Eingangsdaten. So wird ein formales Modell benötigt, welches das Verhalten eines Steuerungssystems hinreichend widerspiegelt. Der Erstellungsprozess der

formalen Modelle erfordert viel Fachwissen und ist fehleranfällig. Darüber hinaus sind formalisierte Anforderungen an das Steuerungssystem erforderlich.

Abbildung 12: Darstellung des Forschungsbedarfs, um formale Verifikationsmethoden für Anlagenbetreiber nutzbar zu machen.

Wie in Abbildung 12 symbolisiert, leitet sich daraus der Forschungsbedarf ab, wie diese Verhaltensmodelle für Anlagenbetreiber automatisiert erstellt werden können. Dies ist vor allem deshalb zunehmend relevant, da sich Software-Änderungen häufig auf das Verhalten eines Steuerungssystems auswirken und somit das Verhaltensmodell angepasst werden müsste. Da zur Anpassung eines komplexen Verhaltensmodells hohe Expertise und Aufwand notwendig sind, ist davon auszugehen, dass aktuelle Verhaltensmodelle selten zur Verfügung stehen. So soll der Ansatz verfolgt werden, dass Verhaltensmodelle für Anlagenbetreiber automatisiert erstellt und bei Funktionsänderungen einfach angepasst werden können.

Die hohe Komplexität der Verhaltensmodelle von Steuerungssystemen und die Vielzahl an funktionalen Anforderungen führen zudem zu einem hohen Verifikationsaufwand. Statt die Effizienz von Verifikationswerkzeugen zu steigern, soll erreicht werden, das betroffene Teilsystem einzugrenzen und somit zielgerichteter Abzusichern.

Wie sich die beschriebene Forschungslücke schließen lässt, wird im folgenden Kapitel beschrieben.

3 Konzept zur Absicherung von verteilten Automatisierungssystemen nach Änderungen der Steuerungssoftware

Ausgehend von der in Kapitel 1 und Kapitel 2 angeführten Einsicht, dass Software-Änderungen an Steuerungssystemen zukünftig häufiger auftreten, wird nachfolgend ein methodischer Ansatz vorgestellt, wie diese Änderungen abgesichert werden können. Da funktionale Änderungen der Steuerungssoftware vor allem in der Betriebsphase öfters auftreten werden, wird die definierte Zielsetzung aus der Perspektive von Anlagenbetreibern verfolgt. Die Analyse des Stands der Forschung ergab, dass Forschungsbedarf in diesem Bereich existiert. Konkret bedarf es systematischer und automatisierter Absicherungsprozesse zur Unterstützung von Anlagenbetreibern nach funktionalen Änderungen der Steuerungssoftware. Einige Aspekte des nun präsentierten Konzepts wurden bereits in folgenden Beiträgen adressiert [126]–[128], eine umfassende Betrachtung steht jedoch bislang aus.

Der Zweck des folgenden Konzepts ist es, Anlagenbetreiber bei der Absicherung nach Software-Funktionsänderungen von Steuerungssoftware zu unterstützen, indem die Vorteile formaler Verifikationsmethoden für sie nutzbar gemacht werden. Formale Verifikationsmethoden können meist vollautomatisiert ausgeführt werden. Im Gegensatz dazu ist die Bereitstellung der dazu notwendigen Eingangsdaten, also der Verhaltensmodelle und funktionalen Anforderungen, anspruchsvoll. Deren Erstellung ist meist mit manuellen Tätigkeiten verbunden, welche fehleranfällig sind und eine gute Systemkenntnis erfordern.

Die dem Konzept zugrundeliegende Idee ist es, diese für Anlagenbetreiber herausfordernde Tätigkeit zu automatisieren. Mithilfe von Modelloperationen sollen, abhängig von der vorangegangenen Software-Änderung, maßgeschneiderte Eingangsdaten für Verifikationswerkzeuge generiert werden. Die Ausarbeitung dieser Konzeptidee wird im Folgenden dargelegt.

3.1 Ansatz zur Ausarbeitung der Konzeptidee

Aufgrund der vollautomatisierten Ausführbarkeit und der Eignung für sicherheitskritische Systeme wird in diesem Konzept das Model Checking, eine formale Verifikationsmethode zur Absicherung von Automatisierungssystemen, verwendet.

Zur Erzeugung der für das Model Checking notwendigen Eingangsdaten, sollen Verhaltensmodelle der Komponenten (Komponentenmodelle) zu einem Verhaltensmodell des Steuerungssystems komponiert, das heißt zusammengefügt werden. Dabei wird die Annahme getroffen, dass

aufgrund modellbasierter Entwicklung und Virtualisierung der Produktion zunehmend Komponentenmodelle zur Verfügung stehen. Die dadurch entstehenden Verhaltensmodelle verteilter Steuerungssysteme sind meist groß. Um der daraus resultierenden Komplexität des betrachteten Systems entgegenzuwirken, sollen vorab von der Funktionsänderung betroffene Teilsysteme mittels einer Auswirkungsanalyse automatisiert erkannt und nur diese gezielt abgesichert werden.

Der Ansatz des entworfenen Konzepts, welches in Abbildung 13 illustriert ist, lässt sich in drei Schritte untergliedern. In diesem Übersichtsbild sind die Komponenten einer verteilten Steuerung als Puzzleteile dargestellt. Diese Puzzleteile beinhalten Verhaltensmodelle, die das Verhalten der Komponenten wiederspiegeln. Nachfolgend wird der Begriff „Komponente“ stellvertretend für Softwarekomponenten verteilter Steuerungen verwendet.

1. Schritt: Mithilfe einer Auswirkungsanalyse wird ermittelt, welche funktionalen Anforderungen (Komponenten-Anforderungen) von Funktionsänderungen betroffen sind und erneut verifiziert werden müssen. Dazu werden anhand der Schnittstellenbeschreibungen der Komponenten die Abhängigkeiten zwischen den Komponenten analysiert. Komponenten, deren funktionale Anforderungen erneut abgesichert werden müssen, sind in Abbildung 13 gelb dargestellt. Die identifizierten Komponenten-Anforderungen bilden den Ausgangspunkt für den zweiten Schritt.

2. Schritt: Komposition der Komponenten, deren Verhaltensmodelle zur Verifikation einer betroffenen Komponenten-Anforderungen notwendig sind. Dazu wird ausgehend von einer betroffenen Komponente (gelbes Puzzleteil) analysiert welche Komponenten zur Erfüllung der Anforderungen beitragen. Wie in Abbildung 13 dargestellt entstehen für die funktionalen Anforderungen aller betroffenen Komponenten individuell komponierte Verhaltensmodelle.

3. Schritt: Das komponierte Verhaltensmodell des betroffenen Teilsystems sowie die dazugehörigen Komponenten-Anforderungen bilden einen maßgeschneiderten Eingang für einen Model Checker, welcher die Gültigkeit der jeweiligen Komponenten-Anforderung überprüft. Dazu sind noch einige Anpassungen am komponierten Verhaltensmodell notwendig.

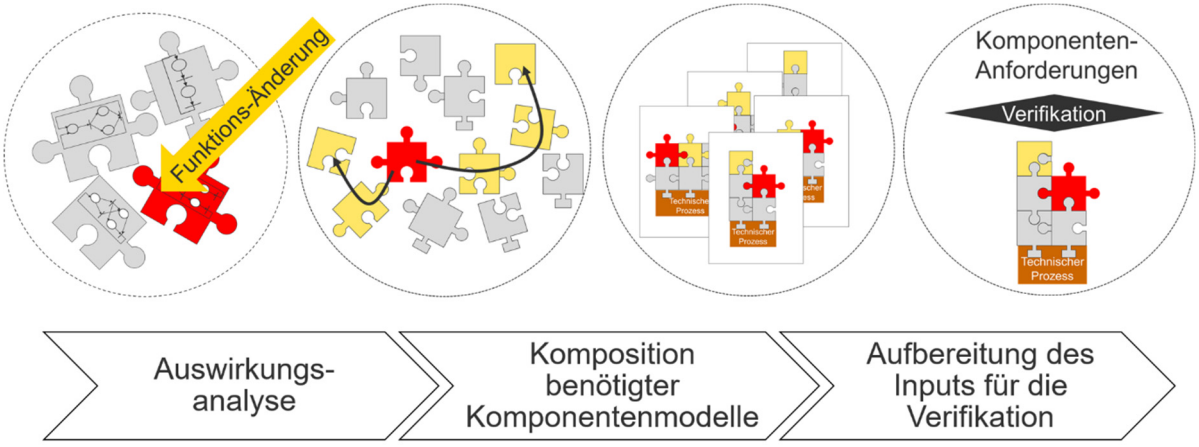


Abbildung 13: Übersicht über die Schritte des Konzepts

Bevor die drei Schritte des Konzepts detailliert erläutert werden, wird grundlegend auf die verwendete Modellierungsart sowie auf die Algorithmen zur Modellkomposition eingegangen. Dazu wird in Kapitel 3.2 ein komponentenbasierter Modellierungsansatz vorgestellt, durch welchen die Komplexität bei der Modellierung reduziert wird. Daraufhin wird die verwendete Modellierungsart zur Verhaltensmodellierung in Kapitel 3.3 beschrieben. In Kapitel 3.4 werden die Mechanismen zur Komposition eines Verhaltensmodells eines größeren Teilsystems aus Verhaltensmodellen der Komponenten (Komponentenmodelle) aufgezeigt.

Auf Basis der in Kapitel 3.4 beschriebenen Mechanismen wird in Kapitel 3.5 dargelegt, wie die Modellierung und die Komposition im Rahmen des dreischrittigen Konzeptes genutzt werden. So wird beschrieben, wie anhand eines automatisiert generierten Abhängigkeitsgraphen von Funktionsänderungen betroffene Komponenten-Anforderungen identifiziert und die zur Verifikation notwendigen Verhaltensmodelle identifiziert, komponiert und adaptiert werden. Am Ende des Konzeptkapitels wird erläutert, wie die komponierten Verhaltensmodelle und die betroffenen Komponenten-Anforderungen zur Verifikation verwendet werden können.

3.2 Komponentenbasierter Modellierungsansatz

3.2.1 Aufgaben beim Aufbaus eines Verhaltensmodells

Verhaltensmodelle ermöglichen die Beschreibung des Verhaltens von Steuerungssystemen. Aufgrund komplexer Steuerungsaufgaben, hohem Funktionsumfang und der starken Vernetzung zwischen den Komponenten sind Verhaltensmodelle von Steuerungssystemen sehr komplex. Um solche Modelle zu erstellen oder bei Funktionsänderungen zu pflegen, ist daher viel Expertise notwendig. Wie in Kapitel 2.3.2.1 erläutert führt dies dazu, dass modellbasierte Verfahren in der Praxis häufig keine Anwendung finden. Zur Vereinfachung der Erstellung von Verhaltensmodellen wird der verteilte Aufbau des Steuerungssystems auf die Modellierung übertragen. Dies erlaubt eine isolierte Modellierung der einzelnen Komponenten.

Die Betrachtung einer isolierten Komponente reicht zur Absicherung funktionaler Anforderungen oftmals nicht aus. Vielmehr ist die Analyse der Interaktion mehrerer Komponenten essentiell. Zur Verifikation dieser Komponenten-Anforderungen muss es deshalb möglich sein, das dazu notwendige Verhaltensmodell aus den Komponentenmodellen des betroffenen Teilsystems zusammenzuführen. Diese Zusammenführung mehrerer Komponentenmodelle wird als Komposition bezeichnet. Dabei muss das komponierte Verhaltensmodell das Verhalten des betroffenen Teilsystems ausreichend widerspiegeln. Der Aufbau der Komponenten, welche als Grundlage der Komposition dienen, wird im Folgenden erläutert.

3.2.2 Nutzung von Modularität zur Reduktion der Komplexität

"The most fundamental problem in software development is complexity. There is only one basic way of dealing with complexity: Divide and conquer." – Bjarne Stroustrup

Die Modularisierung des Verhaltensmodells eines Steuerungssystems analog zu dessen Struktur bietet mindestens drei wesentliche Vorteile:

- reduzierte Komplexität bei isolierter Betrachtung des Modells einer Komponente
- hohe Verfügbarkeit der Verhaltensmodelle von Komponenten, da diese häufig im Rahmen der modellbasierten Entwicklung entstehen
- Wiederverwendung der Modelle von Standardkomponenten

Zur Modellierung der Komponenten wird eine innere und eine äußere Sicht auf die Komponenten definiert. Die innere Sicht, welche das Verhaltensmodell der Komponente beschreibt, wird in Kapitel 3.3.2 erläutert. Das Verhaltensmodell ist gekapselt durch eine äußere Sicht. Die äußere Sicht stellt eine Abstraktion der inneren Sicht dar, indem nur die Schnittstellen einer Komponente, unter Vernachlässigung deren inneren Verhaltens, dargestellt werden. Diese Schnittstellenbeschreibung der äußeren Sicht, welche in Abbildung 14 dargestellt ist, kann automatisiert durch Analyse der inneren Sicht erzeugt werden. Die äußere Sicht wird um Komponenten-Anforderungen ergänzt. Komponenten-Anforderungen sind formalisierte, funktionale Anforderungen, welche an die Funktionalität der jeweiligen Komponenten gestellt werden. Durch diese modulare Definition der Verhaltensmodelle und funktionalen Anforderungen lassen sich die Komponenten isoliert modellieren.

Die in der äußeren Sicht dargestellte Schnittstellenbeschreibung der Komponenten ermöglicht eine Identifikation der Interaktions- und Abhängigkeitsstruktur zwischen den Komponenten. Auf das dazu verwendete Prinzip wird folgend eingegangen.

Wie in Kapitel 2.1.3 beschrieben, verfügen Steuerungskomponenten verteilter Systeme über Fähigkeiten, welche über semantisch definierte Schnittstellen als Services angeboten werden. Durch die semantisch beschriebenen Schnittstellen dieser Fähigkeiten sind Funktionsaufrufe (Call) sowie Bestätigungen einer durchgeführten Funktionalität (Ack) eindeutig definiert und für alle Benutzer interpretierbar. Durch die Analyse dieser Aufrufbeziehungen lassen sich informationstechnische Abhängigkeiten zwischen den Komponenten erkennen. In Abbildung 14 sind diese Kommunikationsschnittstellen rot dargestellt. In dieser generalisierten Darstellung bietet die „Komponente A" die Fähigkeit „ServiceX" an, welche über die Schnittstelle *callServiceX* aufgerufen werden kann. Der Abschluss der aufgerufenen Funktionalität wird über *ackServiceX* zurückgemeldet. Neben Kommunikationsschnittstellen zwischen Komponenten des Steuerungssystems existieren Schnittstellen zu dem technischen Prozess. In Abbildung 14 ist die Beeinflussung des technischen Prozesses durch die Komponente A durch *Aktion* A_j und die Beeinflussung der Steuerung durch

den technischen Prozess durch *Ereignis S_j* in blauer Farbe dargestellt. So kann die Beeinflussung des technischen Prozesses durch Aktoren über Aktionen und die Beeinflussung einer Komponente durch einen Sensor mittels Ereignissen dargestellt werden. Die Farbgebung der Schnittstellentypen wird in dieser Arbeit behalten.

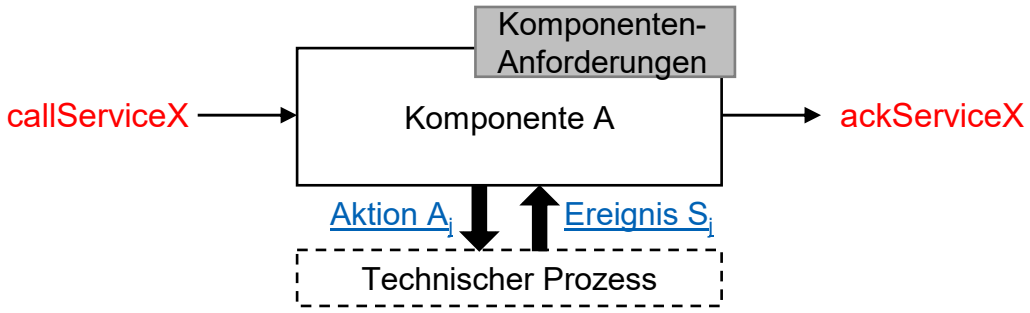


Abbildung 14: Generalisierte Darstellung der äußeren Sicht auf eine Komponente

Analog zu den Verhaltensmodellen werden die funktionalen Anforderungen an das Steuerungssystem modular verwaltet. Somit verfügt jede Komponente über Komponenten-Anforderungen, welche erfüllt sein müssen, um die Funktionalität der Komponente sicherzustellen (siehe Abbildung 14). Aufgrund der Abhängigkeiten innerhalb des Steuerungssystems ist die Erfüllung einer Komponenten-Anforderung nicht nur von der jeweiligen Komponente abhängig, sondern auch von den Komponenten, von welchen diese abhängt. Diese Abhängigkeiten lassen sich über einen Abhängigkeitsgraphen darstellen, dessen automatisierte Generierung in Kapitel 3.5.2.1 beschrieben wird.

Damit die Komponenten-Anforderungen maschinell interpretiert werden können, sind diese in einer formalen Sprache wie beispielsweise CTL formuliert. Bei der Formulierung der Komponenten-Anforderungen wird davon ausgegangen, dass nur die Kenntnis über das Verhalten der jeweiligen Komponente zur Verfügung steht. Somit können sich die Anforderungen initial nur auf Zustände dieser Komponente beziehen. Diese Einschränkung ist notwendig, da zum Modellierungszeitpunkt nicht davon ausgegangen werden kann, dass bekannt ist, mit welchen Komponenten eine Interaktion im Betrieb stattfinden wird.

3.3 Prinzipien der Verhaltensmodellierung

Die Verhaltensmodellierung stellt die innere Sicht des Modularisierungsansatzes dar. Sie beschreibt die Steuerungslogik der Komponenten. Die Kapselung über Schnittstellen ermöglicht eine isolierte Betrachtung des Verhaltensmodells einer Komponente unabhängig von ihrer Umgebung. Dazu wird folgend beschrieben, welchen konzeptionellen Anforderungen die Verhaltensmodellierung genügen muss. Anschließend wird eine geeignete Modellierungsart entworfen.

3.3.1 Konzeptionelle Anforderungen an die Verhaltensmodellierung

In Kapitel 2.2.1.1 wurden zahlreiche Modellierungstechniken zur Beschreibung des Verhaltens der Steuerung eines Automatisierungssystems aufgeführt. Die Anforderungen an die Modellierungsart, welche in Tabelle 4 dargestellt sind, werden nun erläutert.

Tabelle 4: Konzeptionelle Anforderungen an die Verhaltensmodellierung

Konzept-Anforderungen an die Modellierung	
Funktional	**Nicht-Funktional**
• Darstellbarkeit paralleler Prozesse • Abbildbarkeit zeitlichen Verhaltens • Abbildbarkeit ereignisdiskreten Verhaltens • Darstellbarkeit asynchroner Kommunikation • Komponierbarkeit • Abbildbarkeit von Mehrfachzugriff • Formale Definition	• Standardisierung • Hohe Verbreitung • Einfache Modellierbarkeit

Einige Anforderungen an die Modellierung ergeben sich aus den Rahmenbedingungen der Automatisierungstechnik. Die Automatisierungstechnik der diskreten Fertigung ist durch einen hohen Parallelisierungsgrad der Steuerungsprozesse geprägt. Diese Steuerungsprozesse können zeit- sowie ereignisgesteuert ablaufen. Bei der losen Kopplung verteilter, serviceorientierter Systeme kommt der asynchronen Kommunikation zwischen den Komponenten eine große Bedeutung zu [46]. Diese Systeme können Redundanz und Mehrfachzugriff beinhalten. Somit muss die Modellierungsart eine valide Darstellung, von redundanten Komponenten und von Komponenten auf die von mehreren Komponenten zugegriffen werden kann, ermöglichen.

Neben den Anforderungen aus der Domäne Automatisierungstechnik, muss die Modellierungsart einer Reihe konzeptbedingter Anforderungen genügen. Um die Verhaltensmodelle der Komponenten zu einem Teilsystem zusammenfügen zu können, müssen sie komponierbar sein. Eine formale Definition der Modelle ist unabdingbar, damit diese maschinell interpretierbar und verifizierbar sind.

Für die Nutzbarkeit des Konzepts sind neben funktionalen Konzept-Anforderungen auch nichtfunktionale Konzept-Anforderungen an die Modellierungsart relevant. Damit die Modellinformation über ein Austauschformat den Verifikationswerkzeugen übergeben und einheitlich graphisch dargestellt werden kann, ist eine standardisierte Notation der Modelle notwendig. Ein hoher Verbreitungsgrad der Modellierungsart in Forschung und Lehre soll sowohl eine einfache Modellierbarkeit als auch eine möglichst hohe Zugänglichkeit und Anwendbarkeit des Konzepts gewährleisten.

Da zahlreiche Modellierungsarten diesen Anforderungen genügen, können verschiedene Modellierungsarten für das Konzept verwendet werden. Im folgenden Abschnitt wird die für das Konzept entwickelte Modellierungsart beschrieben.

3.3.2 Definition der Modellierungsart

Aufgrund der Erfüllung der Konzept-Anforderungen und ihrer hohen Verbreitung werden Petri-Netze zur Verhaltensmodellierung verwendet. Von Petri-Netzen existieren zahlreiche Variationen, wie NCES, die sich für das Konzept eignen. Zur einfacheren Modellierbarkeit wurde das klassische Petri-Netz nur um diejenigen Aspekte ergänzt, welche zur Beschreibung von verteilten, asynchronen Systemen notwendig sind. Dafür ist eine Erweiterung klassischer Petri-Netze um Schnittstellen erforderlich. Die Kommunikationsschnittstellen zu anderen Komponenten werden mit den Interface-Stellen der „offenen Netze“ ergänzt. Zur Anbindung des technischen Prozesses wird das Petri-Netz um Schnittstellen analog zu SIPN erweitert. Daraus ergibt sich für das erweiterte Petri-Netz N folgendes 8-Tupel:

$$N =< P, T, F, M_0, I, A, S, L, >$$

- $\boldsymbol{P}$: endliche Menge der Stellen p des Netzes.
- $\boldsymbol{T}$: endliche Menge der Transitionen *t* des Netzes.
- $\boldsymbol{F}$: endliche Menge $\mathrm{F}(p,t) \cup F(t,p)$ der Flussrelationen zwischen Stellen und Transitionen. Die Flussrelationen werden über zwei Matrizen dargestellt. Die Menge an Vorgängerstellen einer Transition werden über $\cdot t = \{p|(p,t) \in F\}$ beschrieben. Die Nachfolgerstellen einer Transition werden über $t \cdot = \{p|(t,p) \in F\}$ beschrieben. Entsprechendes gilt für die Vorgängertransitionen einer Stelle $\cdot p = \{t|(t,p) \in F\}$ und Nachfolgertransitionen einer Stelle: $p \cdot = \{t|(p,t) \in F\}$. Die Darstellung der Flussrelationen in zwei getrennten Matrizen $\mathrm{F}(p,t)$ und $F(t,p)$ erlaubt auch die Darstellung nicht-reiner Petri-Netze.

- $\boldsymbol{M_0}$: Startmarkierung, entspricht einem Vektor der Dimension |P|.
- $\boldsymbol{I}$: Interface-Stellen zum Austausch von Informationen mit anderen Komponenten. Dies ist eine Teilmenge der Stellen des Netzes ($I \subseteq P$). Die Bezeichnung der Interface-Stellen entspricht dem Namen der Nachricht, welche über die Schnittstelle ausgetauscht wird. Interface-Stellen können Eingänge (Input-Stellen) oder Ausgänge (Output-Stellen) darstellen. Die Komponentenmodelle stellen minimale Netze dar. Für diese gilt:
  - Stellen I_{Input}, über die eine Nachricht empfangen wird (Input-Stellen), besitzen keine Vorgänger-Transitionen. Somit gilt: $\cdot p = \emptyset$.
  - Stellen I_{Output}, über die eine Nachricht versendet wird (Output-Stellen), besitzen keine Nachfolger-Transitionen. Somit gilt: $p \cdot = \emptyset$.
  - Eine Interface-Stelle I darf nicht gleichzeitig Input und Output sein. Somit gilt: $I_{Output} \cap I_{Input} = \emptyset$.
- **A**: Schnittstelle der Steuerung zum technischen Prozess. Analog zu SIPN werden beim Belegen der Stellen Aktionen auf den technischen Prozess ausgeführt. Dies ist eine Teilmenge der Stellen des Netzes A ⊆ P.
- **S**: Schnittstelle des technischen Prozesses zur Steuerung. Analog zu SIPN werden Transitionen schaltbereit, wenn sich der technische Prozess in einem gegebenen Zustand befindet. Das kann zu einem Ereignis, also einem Zustandsübergang einer Komponente, führen. Dies ist eine Teilmenge der Transitionen des Netzes S ⊆ T.
- $\boldsymbol{L}$: Latenzzeit der Kanten dargestellt über einen Vektor der Dimension $|T|$.

Nachfolgend wird anhand der graphischen Notation auf die Mechanismen des definierten Petri-Netzes eingegangen.

Struktur und Verhalten des Petri-Netzes in Abbildung 15:

Das in Abbildung 15 dargestellte Petri-Netz besteht aus vier Stellen $P = \{P1, Aktion\ A_j, callServiceX, ackServiceX\}$ sowie zwei Transitionen $T = \{T1, Ereignis\ S_j\}$. *P1* und *Aktion A_j* repräsentieren den inneren Zustand der Komponente A. *Aktion A_j* und *Ereignis S_j* beschreiben Schnittstellen zum technischen Prozess (*Aktion $A_j \in A$*; *Ereignis $S_j \in$ S*). *callServiceX* stellt eine Input-Stelle I_{Input} und *ackServiceX* eine Output-Stelle I_{Output} dar. Die Startmarkierung ist $M_0 = [1,0,0,0]^T$. Das Petri-Netz erfüllt dabei immer noch die klassische Petri-Netz-Konvention, dass Stellen ausschließlich mit Transitionen verbunden sind.

Im Ausgangszustand befindet sich Komponente A im Zustand P1. Sobald Service X aufgerufen wird, wird ein Token in der Stelle *callServiceX* erzeugt, wodurch Transition *T1* schaltfähig wird. Durch Schalten der Transition *T1* werden die Token der Vorgängerstellen abgezogen und ein Token in der Nachfolgerstelle *Aktion A_j* erzeugt. Besitzt die Stelle *Aktion A_j* einen Token, wird eine Aktion, beispielsweise durch einen Aktor, auf den technischen Prozess ausgeführt. Tritt *Ereignis S_j* im technischen Prozess, beispielsweise durch ein Sensorwert, auf, wird die entsprechende Transition schaltfähig. Beim Schalten von Transition *Ereignis S_j* wird das Token aus Stelle *Aktion A_j*

abgezogen und jeweils ein Token in Stelle *P1* und Output-Stelle *ackServiceX* erzeugt. Über *ackServiceX* wird durch das Token die Information zurückgegeben, dass die Funktionalität der Komponente A abgeschlossen ist.

Die Bezeichnungen der Schnittstellen zwischen den Komponenten (*callServiceX*, *ackServiceX*) sowie zum und vom technischen Prozess (*Aktion A_j*, *Ereignis S_j*) müssen dabei semantisch definiert sein, um als Grundlage für die Komposition zu dienen. Somit muss die Bedeutung einer Schnittstellenbezeichnung eindeutig definiert sein muss.

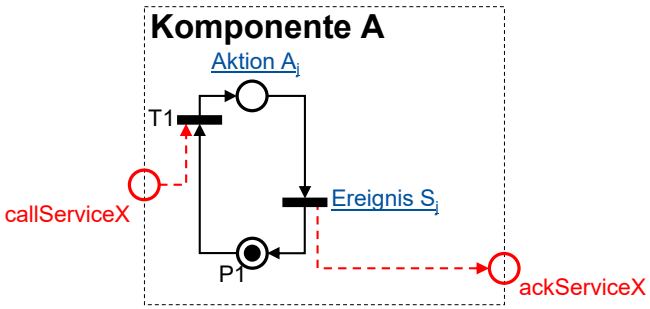


Abbildung 15: Graphische Notation des angepassten Petri-Netzes

Das in Abbildung 15 dargestellte Petri-Netz beschreibt das Verhalten einer beispielhaften Komponente. Um die Interaktion mehrerer Komponenten analysieren zu können, müssen deren Modelle komponiert werden.

3.4 Prinzipien der Komposition von Verhaltensmodellen

Zur Verifikation von funktionalen Anforderungen, welche sich auf mehrere Komponenten beziehen, ist ein Verhaltensmodell notwendig, das das Verhalten des betroffenen Teilsystems widerspiegelt. Dies beinhaltet die Modelle aller Komponenten, die für die Erfüllung der funktionalen Anforderung benötigt werden. Die Prinzipien zur automatischen Generierung des Verhaltensmodells eines Teilsystems aus Verhaltensmodellen einzelner Komponenten, die Komposition, werden im Folgenden betrachtet.

3.4.1 Grundidee für den automatisierten Aufbau der Verhaltensmodelle von Steuerungssystemen

In Kapitel 3.3.1 wurde die Konzept-Anforderung formuliert, dass die gewählte Modellierungsart komponierbar sein muss. Dies bedeutet, dass mehrere Verhaltensmodelle über algebraische Rechenoperationen zu einem Verhaltensmodell zusammengefügt können werden müssen. Die Rechenregeln, welche in Kapitel 3.4.2 beschrieben sind, bilden die Basis für die Komposition der Netze. Zur Automatisierung des Kompositionsvorgangs bedarf es eindeutiger Schnittstellendefinitionen der Komponenten. Dabei wird bei den IT-Schnittstellen der Komponenten ausgenutzt, dass ad-hoc-fähige Komponenten wohldefinierte Schnittstellen besitzen, also syntaktisch sowie

semantisch beschrieben sind. Somit kann deren Semantik als gegeben angenommen werden. Schnittstellen einer Komponente zu dem technischen Prozess müssen ebenso eindeutig definiert sein wie die IT-Schnittstellen. Dann ist die Möglichkeit zur automatisierten Komposition der Modelle gegeben. Dies erlaubt die Erstellung eines zusammengesetzten Verhaltensmodells aus Verhaltensmodellen der Komponenten. Zur Abbildung des Mehrfachzugriffs auf eine Komponente ist eine Erweiterung auf farbige Petri-Netze notwendig. Die automatisierte Umsetzung der farbigen Erweiterung wird in Kapitel 3.4.3 beschrieben.

3.4.2 Rechenregeln zur Komposition von Verhaltensmodellen

Die Komposition von n Komponenten ($N_{Komp.1} \dots N_{Komp.n}$) erfolgt anhand folgender Rechenregeln:

$$P_{ges} = P_{Komp.1} \cup P_{Komp.2} \cup \dots \cup P_{Komp.n}$$
$$T_{ges} = T_{Komp.1} \cup T_{Komp.2} \cup \dots \cup T_{Komp.n}$$
$$F_{ges}(p,t) = F_{Komp.1}(p,t) \oplus F_{Komp.2}(p,t) \oplus \dots \oplus F_{Komp.n}(p,t)$$
$$F_{ges}(t,p) = F_{Komp.1}(t,p) \oplus F_{Komp.2}(t,p) \oplus \dots \oplus F_{Komp.n}(t,p)$$
$$M_{0,ges} = M_{0,Komp.1} \oplus M_{0,Komp.2} \oplus \dots \oplus M_{0,Komp.n}$$
$$I_{ges} = I_{Komp.1} \cup I_{Komp.2} \cup \dots \cup I_{Komp.n}$$
$$A_{ges} = A_{Komp.1} \cup A_{Komp.2} \cup \dots \cup A_{Komp.n}$$
$$S_{ges} = S_{Komp.1} \cup S_{Komp.2} \cup \dots \cup S_{Komp.n}$$
$$L_{ges} = L_{Komp.1} \oplus L_{Komp.2} \oplus \dots \oplus L_{Komp.n}$$

Auf die Mechanismen der Komposition wird nun anhand der graphischen Notation eingegangen.

Struktur und Verhalten des komponierten Petri-Netzes in Abbildung 16:

Die graphische Notation der Komposition zweier Komponenten ist in Abbildung 16 generisch dargestellt. Die Stellen und Transitionen bleiben dabei erhalten. Doppelt vorkommende Stellen-Bezeichnungen, wie sie bei Interface-Stellen häufig auftreten, werden dabei zusammengeführt. Dadurch verbinden sich die einzelnen Netze zu einem zusammenhängenden Netz. Ein komponiertes Netz stellt kein minimales Netz mehr dar, da die Interface-Stellen gleichzeitig Vorgänger- und Nachfolgertransitionen besitzen können (siehe Kapitel 3.3.2).

Der Ausgangszustand des komponierten Petri-Netzes in Abbildung 16 unten ergibt sich aus der Kombination der Ausgangszustände der einzelnen Petri-Netze. Bei Aufruf der Funktionalität von Komponente A durch Komponente B wandert ein Token in die Interface-Stelle *callServiceX*. Dieses Token aktiviert die Transition T_A1 der Komponente A. Nach Durchführung der *Aktion* A_j wird durch Schalten von *Ereignis* S_j ein Token von Komponente A über die Interface-Stelle *ackServiceX* an Komponente B zurückgegeben und somit die Beendigung der Funktionalität bestätigt.

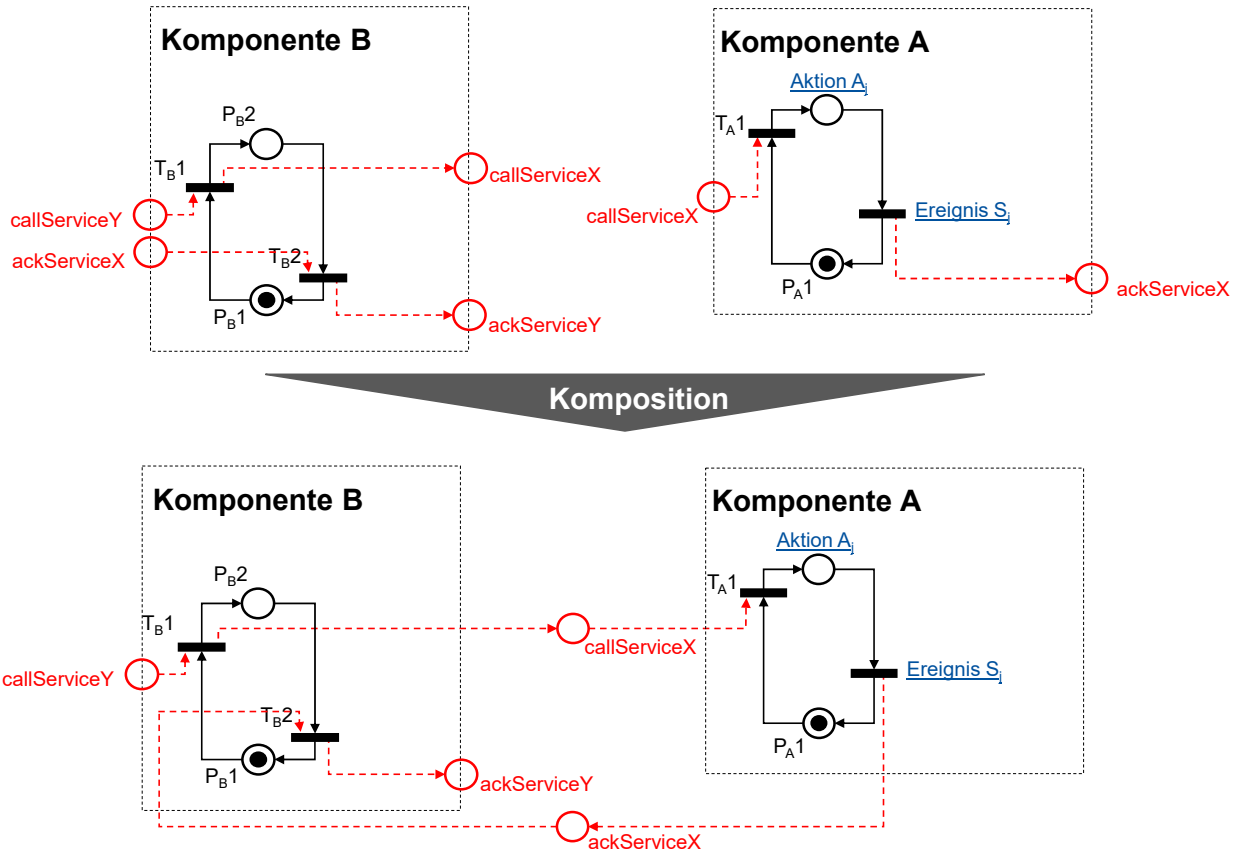


Abbildung 16: Graphische Darstellung der Komposition zweier Komponenten

Die Komposition von Mengen, wie Stellen und Transition, wird dabei durch die Vereinigungsmenge der Elemente der einzelnen Petri-Netze gebildet. Im Gegensatz dazu werden die Matrizen der Flussrelationen und Vektoren der Startmarkierung und Latenzzeiten durch die direkte Summe $\oplus$ komponiert. Die Bildung der direkten Summe ist in Abbildung 17 beispielhaft dargestellt. Dort wird die Komposition der Flussrelationen von Transitionen zu Stellen des Petri-Netzes aus Abbildung 16 mathematisch formuliert. Die direkte Summe spannt die Matrix $F_{ges}(t,p)$ auf, welche alle Verbindungen (Kanten) von Transitionen zu Stellen der komponierten Petri-Netze beschreibt. $F_{ges}(t,p)$ besitzt somit die Dimension $|P_{ges}|x|T_{ges}|$. Existiert dieselbe Interface-Stelle bei mehreren Komponenten, wird sie nur einmalig in die Matrix integriert.

Die Berechnung der Flussrelationen von Stellen zu Transitionen des komponierten Verhaltensmodells $F_{ges}(p,t)$ wird analog dazu durchgeführt.

$$F_{ges}(t,p) = F_B(t,p) \oplus F_A(t,p)$$

$$\begin{array}{r} \\ P_B1 \\ P_B2 \\ P_A1 \\ \text{Aktion } A_1 \\ \text{callServiceX} \\ \text{callServiceY} \\ \text{ackServiceX} \\ \text{ackServiceY} \end{array} \begin{array}{c} \begin{array}{cccc} T_B1 & T_B2 & T_A1 & \text{Ereignis } S_1 \end{array} \\ \begin{pmatrix} \emptyset & 1 & \emptyset & \emptyset \\ 1 & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & 1 \\ \emptyset & \emptyset & 1 & \emptyset \\ 1 & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & 1 \\ \emptyset & 1 & \emptyset & \emptyset \end{pmatrix} \end{array} = \begin{array}{r} \\ P_B1 \\ P_B2 \\ \text{callServiceX} \\ \text{callServiceY} \\ \text{ackServiceX} \\ \text{ackServiceY} \end{array} \begin{array}{c} \begin{array}{cc} T_B1 & T_B2 \end{array} \\ \begin{pmatrix} \emptyset & 1 \\ 1 & \emptyset \\ 1 & \emptyset \\ \emptyset & \emptyset \\ \emptyset & \emptyset \\ \emptyset & 1 \end{pmatrix} \end{array} \oplus \begin{array}{r} \\ P_A1 \\ \text{Aktion } A_1 \\ \text{callServiceX} \\ \text{ackServiceX} \end{array} \begin{array}{c} \begin{array}{cc} T_A1 & \text{Ereignis } S_1 \end{array} \\ \begin{pmatrix} \emptyset & 1 \\ 1 & \emptyset \\ \emptyset & \emptyset \\ \emptyset & 1 \end{pmatrix} \end{array}$$

Abbildung 17: Bildung der direkten Summe der Flussrelationen $F_{ges}(t,p)$ aus $F_B(t,p)$ und $F_A(t,p)$ von Transitionen zu Stellen

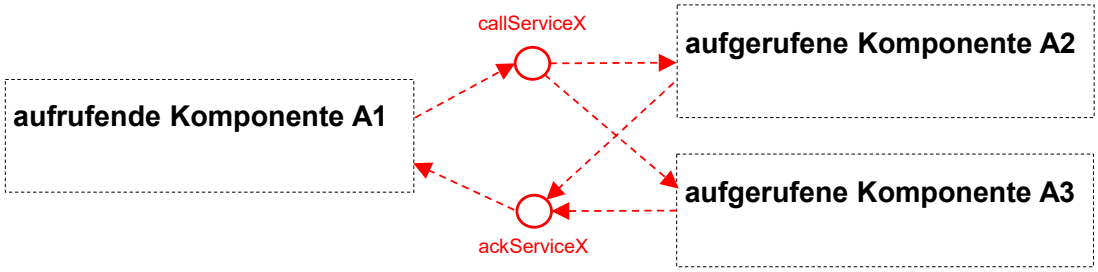
Das in Abbildung 16 dargestellte Beispiel stellt eine einfache 1:1-Beziehung zwischen zwei Komponenten dar. Sind mehrere Komponenten involviert, kann es zu Redundanz und Mehrfachzugriff kommen, was im Folgenden beleuchtet wird.

3.4.3 Berücksichtigung von Redundanz und Mehrfachzugriff

Neben typischen 1:1-Beziehungen zwischen Komponenten kommen in Automatisierungssystemen auch Konfigurationen vor, bei denen mehrere Komponenten die gleiche Funktionalität anbieten (Redundanz) und mehrere Komponenten auf die Funktionalität der gleichen Komponente zugreifen (Mehrfachzugriff). Wie in den Konzept-Anforderungen in Kapitel 3.3.1 beschrieben, müssen diese Eigenschaften bei der Modellkomposition berücksichtigt werden. Im Rahmen des Konzepts wurde ein Ansatz zur Gewährleistung dieser Konzept-Anforderungen entworfen.

Wie aus Abbildung 18 a) ersichtlich wird, stellt die Redundanz bei der Komposition keine Probleme dar. Die Komponenten A2 und A3 aus Abbildung 18 bieten dieselbe Fähigkeit „ServiceX“ an. Durch die Komposition entsteht an der Interface-Stelle *callServiceX* eine Verzweigung, welche die Alternative bei Aufruf der Funktionalität zwischen den beiden Komponenten widerspiegelt. Bei der Interface-Stelle *ackServiceX*, über welche signalisiert wird, dass die Funktionalität abgeschlossen ist, entsteht durch die Komposition eine Zusammenführung der parallelen Pfade. Das Token wandert über die Interface-Stelle *ackServiceX* wieder in die aufrufende Komponente.

a) Redundanz



b) Mehrfachzugriff

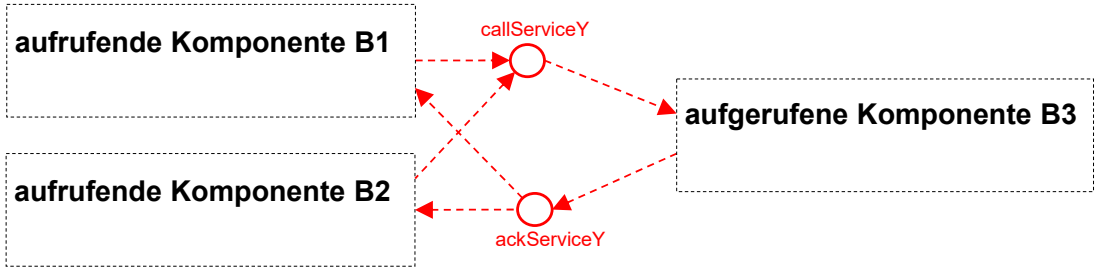


Abbildung 18: Graphische Darstellung der Komposition bei Redundanz (a) und Mehrfachzugriff (b)

Die korrekte Darstellung des Mehrfachzugriffs, abgebildet in Abbildung 18 b), ist anspruchsvoller. Bei Interface-Stelle *callServiceY* entsteht durch die Komposition eine Zusammenführung und bei Interface-Stelle *ackServiceY* eine Verzweigung. Ruft Komponente B1 oder B2 die Funktionalität von Komponente B3 auf, wandert das Token über die Interface-Stelle *callServiceY* in die aufgerufene Komponente B3. Bei Beendigung der Funktionalität wandert das Token in die Interface-Stelle *ackServiceY*. Ausgehend von dieser Stelle führen zwei Kanten zu jeweils einer der Komponenten. Wird das Token von der Komponente abgezogen, welche nicht die Funktionalität aufgerufen hat, kann dies zu einem Verhalten führen, welches nicht das Verhalten des Steuerungssystems widerspiegelt.

Um das Abziehen des Tokens durch die unbeteiligte Komponente zu verhindern, wurde ein Konzept entworfen, mit welchem betroffene Modelle automatisiert zu farbigen Petri-Netzen erweitert werden. Dies lässt eine eindeutige Zuordnung des Tokens zu der aufrufenden Komponente zu. Die folgend beschriebene farbige Erweiterung der Petri-Netze ist immer dann notwendig, wenn Mehrfachzugriff besteht und eine Bestätigung *(Ack)* an die aufrufende Komponente zurückgesendet wird. Bei Komponenten, die nur aufgerufen werden und keine Bestätigung zurücksenden, ist die Erzeugung von farbigen Token auch bei Mehrfachzugriff nicht notwendig, da die Mehrdeutigkeit erst bei der Bestätigung entsteht. Die farbige Erweiterung der Petri-Netze im Falle eines Mehrfachzugriffs ist in Abbildung 19 dargestellt.

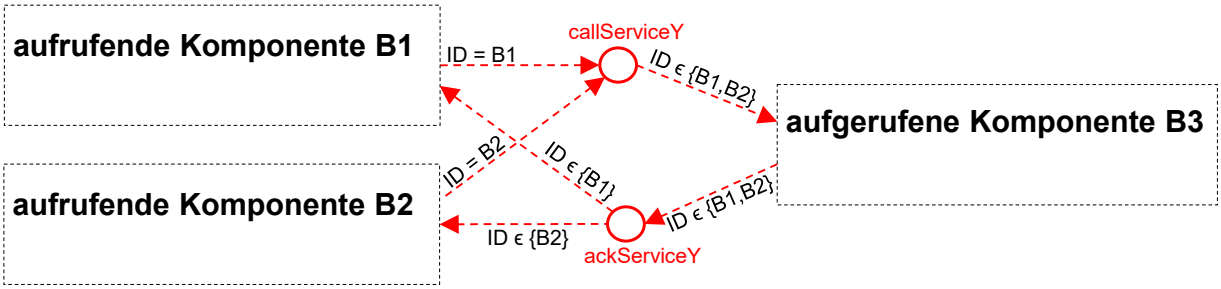


Abbildung 19: Funktionsweise zur Koordination des Mehrfachzugriffs durch Erweiterung zu farbigen Petri-Netzen

Struktur und Verhalten des farbigen Petri-Netzes aus Abbildung 19:

In Abbildung 19 ist das Prinzip dargestellt, nach dem die Schnittstellen zwischen den Verhaltensmodellen beteiligten Komponenten farbig erweitert werden. Ruft Komponente B1 die Funktionalität von Komponente B3 auf, wird ein Token mit der Farbe ID=B1 an die Interface-Stelle *callServiceY* übergeben. Diese Interface-Stelle repräsentiert den Funktionsaufruf der Komponente B3. Bei Funktionsdurchführung wandert das Token der Farbe ID=B1 durch das Petri-Netz der Komponente B3 bis zur Interface-Stelle *ackServiceY*, über welche die Bestätigung der Funktionsdurchführung an die aufrufende Komponente zurückgesendet wird.

Anhand der Farbe des Tokens ist an Interface-Stelle *ackServiceY* nun unterscheidbar, von welcher Komponente die Funktionalität der Komponente B3 aufgerufen wurde. Damit nun das Token ausschließlich an die aufrufende Komponente B1 gesendet wird, verfügen die Nachfolger-Transitionen von *ackServiceY* über Wächter (engl. guards). Diese Wächter befinden sich in Komponente B1 und Komponente B2 und sind deshalb nicht in Abbildung 19 dargestellt. Sie lassen das Schalten nur zu, wenn die Farbe des Tokens der Bezeichnung der jeweiligen Komponente entspricht. So ist die Transition der aufrufenden Komponenten B1 nur für einen Token der Farbe „B1“ schaltfähig und der mögliche Pfad innerhalb des Modells eindeutig.

Damit die Farbinformation des Tokens vom Funktionsaufruf bis zur Bestätigung des Abschlusses der Funktionalität durch das Petri-Netz transportiert wird, findet nach der Komposition eine automatisierte Analyse der mehrfach aufrufbaren Komponenten statt. Dabei wird, über eine Breiten- oder Tiefensuche, analysiert, welche Pfade vom Aufruf bis zur Beendigung der Funktionalität führen. Damit die Kanten, die auf den identifizierten Pfaden liegen, die Farbinformation an die Nachfolgerstelle weitergeben, werden die Flussrelationsmatrizen um die Farbvariablen automatisiert erweitert. Abbildung 20 stellt die Detaillierung von Abbildung 19 dar. Durch die Transition *T1* der Komponente B1 wird ein farbiges Token (ID=B1) erzeugt. Die darauffolgenden Kanten transportieren die Farbinformation weiter. Dies ist durch das Kantenattribut ID dargestellt. An Transition *T2* der Komponente B1 befindet sich ein Guard [ID=B1], welcher dafür sorgt, dass die Transition nur schaltet, wenn das Token die Farbe ID=B1 besitzt. Durch Schalten der Transition T2 geht die Farbinformation wieder verloren, da die darauffolgende Kante das Attribut ID nicht trägt.

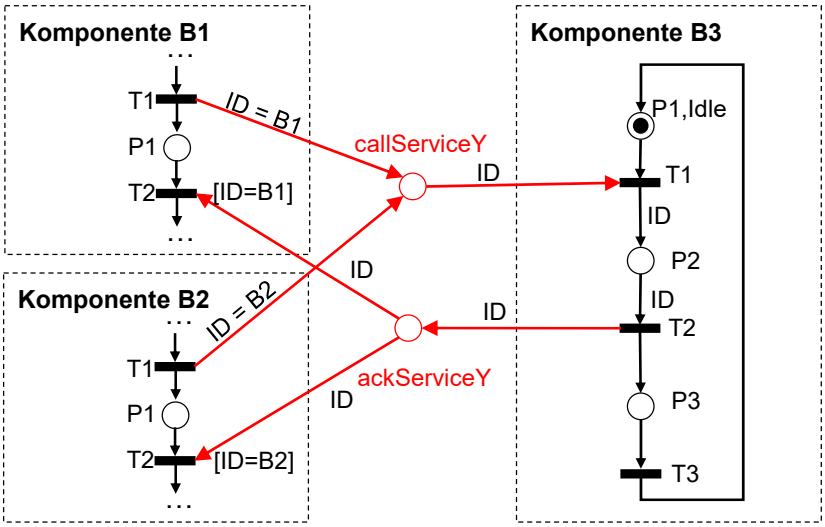


Abbildung 20: Farbige Erweiterung einer von mehreren Komponenten aufrufbaren Komponente

Eine aufgerufene Komponente kann ihrerseits weitere Komponenten aufrufen. Dies ist graphisch in Abbildung 21 veranschaulicht. Aufgrund eines möglichen Mehrfachzugriffs muss die Komponente B3 farbig erweitert werden. Das farbige Token muss dabei nicht über mehrere Aufruf-Ebenen an Komponente B4 weitergegeben werden. Dies ist in der Annahme begründet, dass eine Komponente zeitgleich nur einen Client bedienen kann. Dadurch ist der Aufruf einer weiteren Komponente (Komponente 4) stets eindeutig zuordenbar. Um die Eindeutigkeit zu gewährleisten, wäre es, ohne Treffen dieser Annahme, notwendig, das farbige Token an die folgend aufgerufenen Komponente B4 weiterzureichen. Dies müsste mit einer enormen Steigerung der Komplexität des Petri-Netzes erkauft werden.

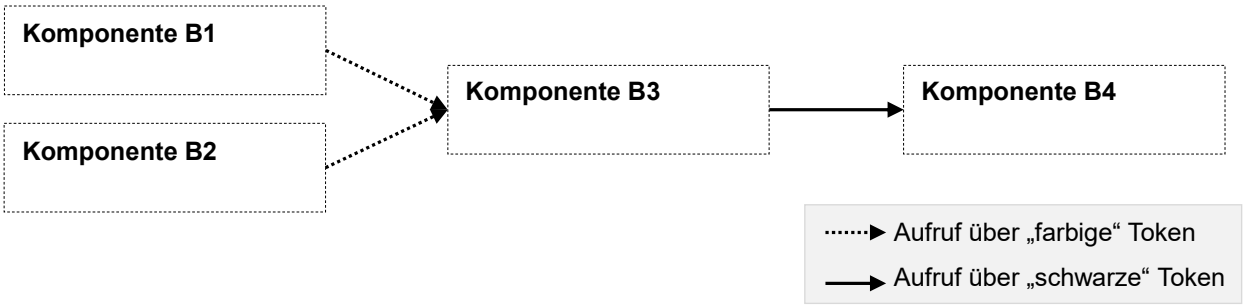


Abbildung 21: Darstellung der farbigen Erweiterung bei Aufruf über mehrere Ebenen

Ist bei Komponente B4 ebenfalls Mehrfachzugriff möglich, erfolgt die Erzeugung des farbigen Tokens analog zu Abbildung 19.

3.4.4 Modellierung und Anbindung des technischen Prozesses

Im Fokus des Konzepts stehen Funktionsänderungen an der Steuerungssoftware. Da Abhängigkeiten des Steuerungssystems auch durch den technischen Prozess bedingt sein können, muss dieser ebenfalls bei in Modellierung berücksichtigt werden. Darüber hinaus lassen sich über den technischen Prozess Sicherheitsanforderungen definieren, worauf in Kapitel 3.5.4.2 eingegangen wird.

Der technische Prozess stellt kein zu testendes Modul dar. So genügt es, ihn, wie beim Software-Integrationstest üblich, als Platzhalter (engl. stub) zu modellieren. Platzhalter beschreiben beim Softwaretest Pseudo-Module, die mit dem Testobjekt interagieren und partiell das Verhalten der jeweiligen Module beschreiben, mit welchen eine Interaktion stattfindet [4]. Bei der Platzhaltermodellierung ist es ausreichend, die Module rudimentär zu beschreiben. So genügt es, die Darstellung des technischen Prozesses auf die Modellierung essentieller Abhängigkeiten zwischen Aktorik und Sensorik zu beschränken.

Zur Modellierung des technischen Prozesses werden prozessbezogen interpretierte Petri-Netze (PIPN) verwendet. PIPN können analog zu klassischen, zeitbehafteten Petri-Netzen über folgendes 5-Tupel beschrieben werden:

$$N_{PIPN} =< P, T, F, M_0, L >$$

Die graphische Notation des PIPN ist in Abbildung 22 dargestellt. Das dargestellte prozessbezogen interpretierte Petri-Netz besteht aus drei Stellen ($P = \{P1, P2, Aktion\ A_j\}$) mit der Startmarkierung $M_0 = [1,0,0]^T$ sowie drei Transitionen ($T = \{T1, T2, Ereignis\ S_j\}$), wovon zwei zeitbehaftet sind ($L = [t_1, t_2, 0]^T$). *P1* und *P2* geben den Zustand des technischen Prozesses wieder. Dies kann beispielsweise der aktuelle Standort eines Werkstücks sein. *Aktion* A_j und *Ereignis* S_j repräsentieren Schnittstellen zur Steuerung. Diese Schnittstellen finden sich in Steuerungskomponenten wieder. Wird die Stelle *Aktion* A_j in einer Steuerungskomponente durch ein Token belegt, wird dadurch ein Aktor aktiviert, welcher eine Aktion auf den technischen Prozess ausführt. Nach Ablauf der definierten Zeitdauer t_1 schaltet die Transition *T1* und ein Token wandert von *P1* zu *P2*. Durch Belegen der Stelle *P2* ist die Transition *Ereignis* S_j schaltfähig. Dies könnte beispielsweise ein Sensor sein, der ein Werkstück an Position *P2* detektiert. So können die Steuerungskomponenten, welche die Transition *Ereignis* S_j besitzen, durch den technischen Prozess beeinflusst werden.

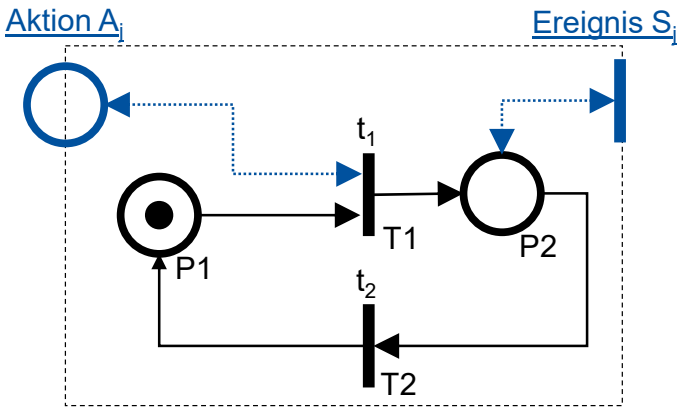


Abbildung 22: Graphische Notation des prozessbezogen interpretierten Petri-Netzes, welches als Platzhalter den technischen Prozess beschreibt

Die Kommunikation zwischen einer Komponente und dem technischen Prozess kann nicht auf gleiche Art und Weise modelliert werden wie die Kommunikation zwischen Komponenten. Bei der Kommunikation zwischen Komponenten werden Nachrichten von einem Absender zu einem Empfänger gesendet, welche über einen Token-Austausch abstrahiert dargestellt werden. Die Schnittstelle zwischen einer Komponente und dem technischen Prozess basiert auf den aktuellen Systemzustand ohne Austausch von Nachrichten. Dies kann analog zu der Verbindung von SIPN und PIPN durch Verwendung von Testkanten realisiert werden [129]. Die Testkanten sind über einen Doppelpfeil symbolisiert. Diese bewirken, dass die Nachfolgertransition, analog zu klassischen Kanten, nur schaltfähig ist, wenn die Vorgängerstelle belegt ist. Im Gegensatz zu klassischen Petri-Netzen wird der Vorgängerstelle der Testkante dabei kein Token abgezogen. Dies führt dazu, dass bei den Schnittstellen zum bzw. vom technischen Prozess (*A* bzw. *S*) kein Token durch Schalten der Transitionen abgezogen wird. Andernfalls würde dies den Zustand des technischen Prozesses und der Steuerung verfälschen. So würde bei Schalten der Transition *Ereignis* S_1 das Token, welches den Zustand des technischen Prozesses beschreibt, durch die Schnittstelle von der Stelle *P2* zu einer Steuerungskomponente, fälschlicherweise abgezogen werden. Somit wäre der technische Prozess zustandslos.

Wie beschrieben ist die mathematische Notation des klassischen Petri-Netzes auch für die Beschreibung des PIPN ausreichend. Da bei Testkanten kein Token übertragen wird, ist zu beachten, dass bei den Matrizen der Flussrelationen *F(p,t)* und *F(t,p)* zwischen einer Null und der leeren Menge unterschieden wird. Die leere Menge bedeutet, dass keine Verbindung zwischen einer Stelle und Transition existiert. Die Null bedeutet, dass zwar eine Verbindung existiert, aber aufgrund des Vorhandenseins einer Testkante kein Token beim Schalten der Transition hinzugefügt oder abgezogen wird.

Eine farbige Erweiterung des technischen Prozesses ist nicht notwendig, da es sich nicht um einen Funktionsaufruf handelt, bei welchem eine Rückmeldung an die aufrufende Komponente gegeben wird.

3.5 Prozess zur Identifikation und Verifikation von Änderungen betroffener Teilsysteme

Um bei Funktionsänderungen einer Komponente nicht das gesamte Steuerungssystem erneut absichern zu müssen, können deren Auswirkungen auf andere Komponenten analysiert werden. Durch solch eine Auswirkungsanalyse werden jene Anforderungen der Komponenten identifiziert, deren Gültigkeit aufgrund der Softwareänderung nicht mehr gewährleistet ist. Gegen die übrigen nicht betroffenen Komponenten-Anforderungen muss folglich nicht verifiziert werden.

Zur Verifikation einer betroffenen Komponenten-Anforderung wird meist nicht das Verhaltensmodell des gesamten Steuerungssystems benötigt. Indem nur das Verhaltensmodell des notwendigen Teilsystems komponiert wird, welches für die Verifikation einer Komponenten-Anforderung notwendig ist, lassen sich die Größe und Komplexität des daraus resultierenden Verhaltensmodells stark reduzieren. Auf die dazu notwendigen Schritte wird im Folgenden eingegangen.

3.5.1 Aufgaben der Absicherung betroffener Teilsysteme

Aufbauend auf der beschriebenen Modellierungsart wird, ausgehend von einer Softwareänderung, ein maßgeschneiderter Eingang für Model Checker generiert. Diese Eingangsdaten bestehen aus Komponenten-Anforderungen und den zugehörigen komponierten Petri-Netzen. Die dazu notwendigen Schritte sind in der Übersicht des Konzepts zu Beginn von Kapitel 3 (Abbildung 13) illustriert. In Kapitel 3.5.2.1 wird beschrieben, wie der Aufbau des Abhängigkeitsgraphen auf Basis der Komponentenmodelle erfolgt. Anschließend wird in Kapitel 3.5.2.2 erläutert, wie auf Basis des Abhängigkeitsgraphen eine Auswirkungsanalyse durchgeführt werden kann. Bei der Auswirkungsanalyse werden von Softwareänderungen betroffene Komponenten-Anforderungen identifiziert. Zur Verifikation einer Komponenten-Anforderung muss die Interaktion mehrerer Komponentenmodelle betrachtet werden können. Die Komposition von Komponentenmodellen, die zur Absicherung einer betroffenen Komponenten-Anforderungen notwendig sind, wird in Kapitel 3.5.3 aufgezeigt. Schließlich wird in Kapitel 3.5.4 beschrieben, wie die Eingangsinformationen für den Model Checker angepasst und funktionale Anforderungen definiert werden.

3.5.2 Schritt 1: Auswirkungsanalyse

3.5.2.1 Aufbau des Abhängigkeitsgraphen eines Steuerungssystems

Funktionale Abhängigkeiten innerhalb des Steuerungssystems entstehen durch Aufruf der Funktionalität einer Komponente durch eine andere. Da die Funktionsaufrufe semantisch beschrieben sind, können diese anhand der Bezeichnung der Interface-Stellen automatisiert erkannt und interpretiert werden. Bei den minimalen Netzen der einzelnen Komponenten sind Output-Stellen dadurch gekennzeichnet, dass sie keine Nachfolgertransitionen besitzen. Input-Stellen besitzen

dagegen keine Vorgängertransitionen. Anhand der semantischen Beschreibung lässt sich erkennen, ob es sich bei der Schnittstelle um einen Funktionsaufruf (Call) oder um die Bestätigung des Abschlusses einer Funktion (Ack) handelt. Zum Aufbau des Abhängigkeitsgraphen sind nur die Funktionsaufrufe relevant. Bei einem Funktionsaufruf beschreibt eine Output-Stelle den Aufruf einer Funktionalität und eine Input-Stelle das Anbieten einer Funktionalität.

Die extrahierten Aufrufinformationen der Komponenten lassen sich anschaulich mittels der äußeren Sicht der Komponenten darstellen. Die Auflösung dieser Aufrufinformationen ermöglicht den automatisierten Aufbau eines Abhängigkeitsgraphen, welcher die Abhängigkeiten zwischen den Komponenten des Steuerungssystems widerspiegelt. Zur graphischen Darstellung eines Abhängigkeitsgraphen eignet sich das Blockdefinitionsdiagramm sehr gut. Weitergehend bietet es den Vorteil, dass es in der Modellierungssprache SysML standardisiert ist, welche in der Automatisierungstechnik verbreitet ist. Die Komponenten werden über Rechtecke dargestellt (siehe Abbildung 23). Der Ausgangspunkt eines Pfeils stellt einen Funktionsaufruf der jeweiligen Komponente dar. Der Endpunkt des Pfeils stellt das Anbieten einer Funktionalität der entsprechenden Komponente dar. Bei Redundanz und Mehrfachzugriff von Komponenten muss jede Kombinationsmöglichkeit durch einen Pfeil abgebildet werden. So greifen in Abbildung 23 Komponente A und Komponente B auf die Funktionalität von Komponente D zu. Die Anforderungen, die an eine Komponente gestellt werden, sind zusätzlich am oberen rechten Eck der Komponenten angeheftet. Bei hierarchisch verteilten Systemen entsteht ein gerichteter azyklischer Graph. Wie anhand dieses Graphen abgeleitet werden kann, welche Komponenten-Anforderungen durch Funktionsänderungen betroffen sind und welche Komponenten zur Absicherung von Komponenten-Anforderungen notwendig sind, wird nachfolgend beschrieben.

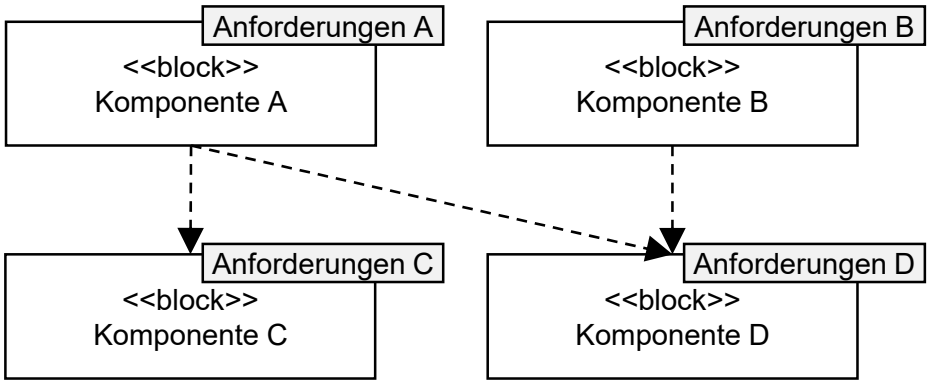


Abbildung 23: Blockdefinitionsdiagramm zur Darstellung von Abhängigkeiten. Es bestehen Abhängigkeiten von Komponente A zu Komponente C und D sowie von Komponente B zu Komponente D

3.5.2.2 Auswirkungsanalyse zur Identifikation zu verifizierender Komponenten-Anforderungen

Nach Kapitel 3.2.2 besitzt jede Komponente eine oder mehrere Komponenten-Anforderungen, welche erfüllt sein müssen, um die Funktionalität der Komponente zu gewährleisten. Dies ist in Abbildung 23 dargestellt. Wird eine Komponente geändert, kann die Gültigkeit ihrer Komponenten-Anforderungen nicht mehr gewährleistet werden. Sind noch weitere Komponenten von der geänderten Komponente abhängig, kann die Gültigkeit von deren Komponenten-Anforderungen ebenso nicht mehr gewährleistet werden. Somit müssen auch deren Komponenten-Anforderungen erneut abgesichert werden.

Dies ist anhand des Blockdefinitionsdiagramms in Abbildung 24 dargestellt. So müssen alle Komponenten, die entgegen der Pfeilrichtung der Abhängigkeitspfeile von der geänderten Komponente erreichbar sind, erneut abgesichert werden. Findet in Komponente E eine Funktionsänderung statt, müssen folglich die funktionalen Anforderungen der Komponenten A, C und E erneut abgesichert werden.

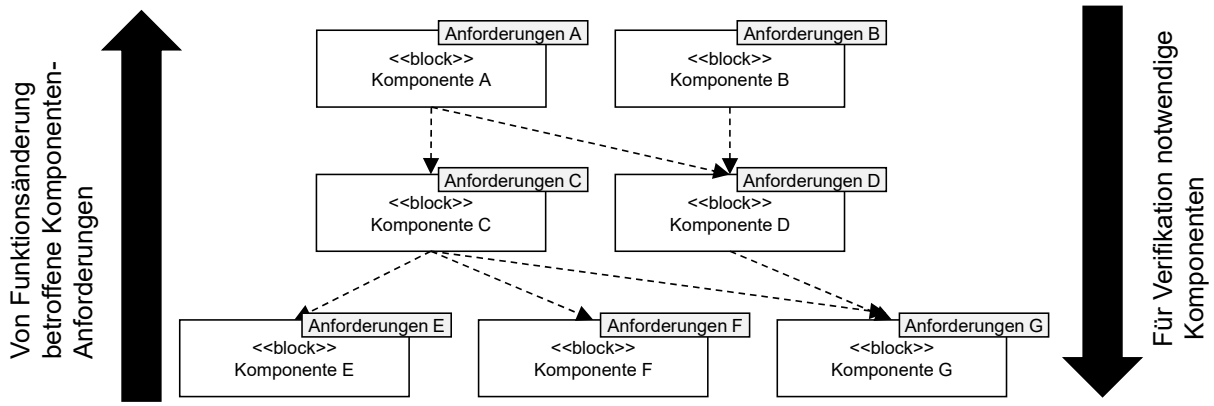


Abbildung 24: Vorgehen bei der Identifikation betroffener Komponenten-Anforderungen und zur Verifikation notwendiger Komponenten

3.5.3 Schritt 2: Komposition zur Verifikation benötigter Komponentenmodelle

Um Rechenaufwand und Komplexität zu reduzieren, werden nur die Petri-Netze komponiert, welche zur Verifikation der Anforderungen einer Komponente notwendig sind.

Ist eine geänderte Komponente von anderen abhängig, kann die Erfüllung ihrer Funktionalität nur in Abhängigkeit der Komponenten angegeben werden, von welchen diese abhängig ist. Wie in Abbildung 24 dargestellt, müssen somit zur Absicherung der funktionalen Anforderungen einer Komponente alle Komponenten komponiert werden, welche im Blockdefinitionsdiagramm in Pfeilrichtung von der jeweiligen Komponente erreichbar sind. Zur Absicherung der funktionalen Anforderungen von Komponente C müssen somit die Verhaltensmodelle der Komponenten C, E, F und G komponiert werden. Dies geschieht nach den in Kapitel 3.4.2 dargelegten Rechenregeln.

Um Mehrfachzugriff zu berücksichtigen wird das komponierte Verhaltensmodell anhand des in Kapitel 3.4.3 beschriebenen Vorgehen farbig erweitert, sobald die beschriebenen Gegebenheiten für Mehrfachzugriff erfüllt sind. Das ist in dem eben genannten Beispiel nicht der Fall, da bei dem komponierten Petri-Netz nur von Komponente C auf die Komponenten E, F, und G zugegriffen wird.

3.5.4 Schritt 3: Aufbereitung der Eingangsinformationen für die formale Verifikation

Zur Verifikation des komponierten Verhaltensmodells bedarf es einiger Anpassungen. So erfolgt eine Erweiterung des Verhaltensmodells und eine Einbindung der funktionalen Anforderungen, gegen die verifiziert wird. Die funktionalen Anforderungen sowie das erweiterte Verhaltensmodell bilden die Eingangsinformationen eines Model Checker, welcher anschließend automatisiert das Verifikationsergebnis berechnet.

3.5.4.1 Erweiterung des komponierten Verhaltensmodells

Die Komponenten-Anforderungen, welche verifiziert werden sollen, beziehen sich auf die Funktionalität einer Komponente. Die Input-Stelle, über welche die Funktionalität aufgerufen wird, stellt den Startpunkt bei der Ausführung der Funktionalität dar. Durch Hinzufügen eines zusätzlichen Token in diese Stelle wird der Initialzustand des Systems bei Aufruf der Funktionalität im Verhaltensmodell abgebildet.

In Schritt 2 wurden ausschließlich die Verhaltensmodelle der Komponenten komponiert. Die Anbindung des technischen Prozesses geschieht anschließend nach dem in Kapitel 3.4.4 beschriebenen Verfahren. Welche Modelle des technischen Prozesses zur Verifikation einer Komponenten-Anforderung notwendig sind, ist gemeinsam mit den Komponenten-Anforderungen definiert, deren Inhalt im Folgenden erläutert wird.

3.5.4.2 Definition der funktionalen Anforderungen

Bei funktionalen Anforderungen wird zwischen Komponenten-Anforderungen und Sicherheitsanforderungen des technischen Prozesses unterschieden. Komponenten-Anforderungen beziehen sich auf das Verhalten der betrachteten Komponente und können deshalb während der Entwicklungszeit formuliert werden, ohne dass der spätere Anwendungskontext bekannt ist. Sicherheitsanforderungen beziehen sich auf einen konkreten Anwendungskontext, zum Beispiel eine spezifische Produktionsanlage. Somit müssen diese anwendungsspezifisch definiert werden.

Komponenten-Anforderungen

Zur Spezifikation der Komponenten-Anforderungen werden folgende Informationen in formalisierter Form benötigt:

- Anforderungen, die an die Funktionalität der Komponente gestellt werden.
- Angabe, welche Module des technischen Prozesses notwendig sind
- Anfangszustand des technischen Prozesses

Somit sind zur Spezifikation der Komponenten-Anforderungen, neben den formalisierten, funktionalen Anforderungen, weitere Informationen notwendig. Die formalisierten Komponenten-Anforderungen sollten bei der Erstellung des Verhaltensmodells einer Komponente aufgestellt werden, da üblicherweise nur der Komponentenentwickler ausreichend Kenntnis über den Aufbau des Verhaltensmodells besitzt. Sie bilden Soll-Eigenschaften ab, gegen die der Model Checker später verifizieren kann. Somit müssen diese in einer formalen, maschineninterpretierbaren Form, wie CTL, vorliegen. Anhand der Komponenten-Anforderungen lassen sich unter anderem folgende Soll-Eigenschaften spezifizieren, welche die Komponente besitzen soll:

- Lebendigkeit
- Reversibilität
- Erreichbarkeit (nicht-) erlaubter Zustände
- Erreichbarkeit (nicht-) erlaubter Zustandssequenzen

Funktionale Anforderungen, welche an Komponente B aus Abbildung 25 gestellt werden können, sind beispielhaft in Tabelle 5 dargestellt.

Tabelle 5: Funktionale Komponenten-Anforderungen

CTL-Anweisung	**Beschreibung**
EX (P2)	Die Stelle P2 muss von der Startmarkierung im nächsten Schritt erreichbar sein.
AF (P1)	Stelle P1 muss von jedem Folgezustand aus immer erreichbar.
EF(ackServiceX)	Es gibt mindestens einen, von der Startmarkierung ausgehender Pfad, auf dem die Stelle ackServiceX erreichbar ist.

Bei der Modellierung und Spezifikation einer Komponente ist nicht bekannt, mit welchen Komponenten diese im Betrieb interagiert. So wird angenommen, dass zum Zeitpunkt der Erstellung der Komponenten-Anforderungen nur der Aufbau der jeweiligen Komponente bekannt ist. Das

resultiert darin, dass sich die Komponenten-Anforderung ausschließlich auf das Verhalten dieser Komponente beziehen kann. In Abbildung 25 ist dies veranschaulicht. So ist es nicht möglich, verbotene Zustände oder geforderte Zustandssequenzen einer anderen Komponente zu definieren.

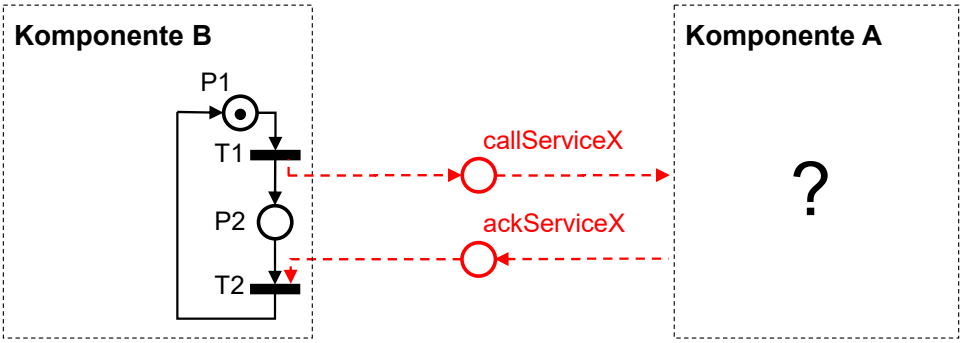


Abbildung 25: Bekanntes Systemverhalten beim Entwurf der Anforderungen an die Komponente B

Definition von Sicherheitsanforderungen an den technischen Prozess

Nimmt der technische Prozess einen verbotenen Zustand an, kann dies sicherheitskritische Auswirkungen haben. Dadurch existieren im Gegensatz zu rein informationstechnischen Systemen bei Steuerungen von Automatisierungssystemen weitere Abhängigkeiten, welche bei der Modellierung berücksichtigt werden müssen [91]. Zur Überprüfung, ob verbotene Zustände des technischen Prozesses eingenommen werden können, gibt es zwei Möglichkeiten – die Modellierung der verbotenen Zustände als formalisierte Anforderung oder die Modellierung der verbotenen Zustände als totale Verklemmung direkt im Modell des technischen Prozesses:

1. Formale Anforderungen zur Beschreibung verbotener Zustände: Ergänzung der Komponenten-Anforderungen um Anforderungen des technischen Prozesses, die beschreiben, welche Zustände nicht eingenommen werden dürfen. Dies muss anwendungsspezifisch durchgeführt werden.

2. Änderung des PIPN zur Beschreibung verbotener Zustände: Dazu wird das prozessbezogen interpretierte Petri-Netz um Stellen und Transitionen erweitert. Wird ein verbotener Zustand eingenommen, wird eine hinzugefügte Transition schaltfähig, welche die Token des technischen Prozesses von den Stellen, die den verbotenen Zustand darstellen abzieht und in einer totalen Verklemmung resultiert. In Abbildung 26 ist dies beispielhaft dargestellt. Es existiert ein verbotener Zustand, wenn die Stellen *P2* und *P3* zeitgleich belegt sind. Aufgrund des integrierten Sicherheitsmechanismus werden die Token über die Transition TD_1 abgezogen. Dadurch entsteht im technischen Prozess eine totale Verklemmung, welche zur totalen Verklemmung der Steuerung führt. Bei systematischer Benennung der totalen Verklemmungen, lässt sich darüber hinaus gezielt nach Erreichbarkeit der Stellen durch das Model Checking prüfen.

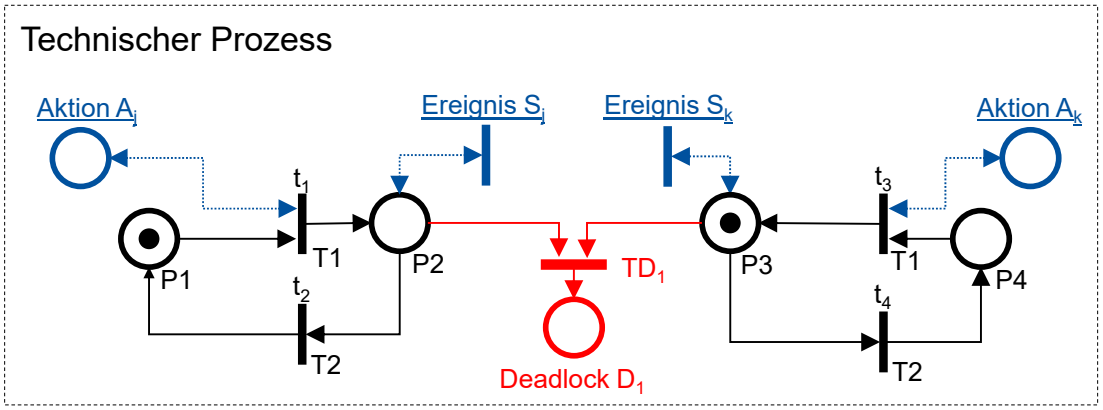


Abbildung 26: Erweiterung eines prozessbezogen interpretierten Petri-Netzes zur Definition verbotener Zustände

3.5.4.3 Verifikation des betroffenen Teilsystems

Zu Ende des in Kapitel 3.5 beschriebenen „Prozesses zur Identifikation und Verifikation von Änderungen betroffener Teilsysteme“ stehen die notwendigen Eingangsdaten für das Model Checking bereit. Für jede Komponente, deren Anforderungen gemäß der Auswirkungsanalyse, nicht mehr gewährleistet werden können, sind dies:

- Ein komponiertes Verhaltensmodell in einem definierten Startzustand. Dies besteht aus Verhaltensmodellen der Steuerungskomponenten sowie des technischen Prozesses.
- Alle funktionalen Anforderungen, welche an die abzusichernde Komponente gestellt werden.

Für jede betroffene Komponente ist anschließend ein separates Model Checking möglich, um die Gültigkeit ihrer funktionalen Anforderungen zu beweisen. Als Ergebnis des Model Checking steht somit die Aussage, ob die funktionalen Anforderungen nach Funktionsänderungen noch erfüllt sind. Dabei wurden gezielt nur die funktionalen Anforderungen überprüft, deren Gültigkeit nach einer Funktionsänderung nicht mehr gewährleistet werden konnten.

4 Realisierung des Konzepts

Die Funktionsweise und Anwendbarkeit des Konzepts wurde anhand einer konkreten Implementierung demonstriert. TestIAS ist ein Testgerät zur Absicherung von Automatisierungssystemen nach Änderung der Steuerungssoftware. TestIAS ermöglicht eine für den Anlagenbetreiber vollautomatisierte Überprüfung, ob alle funktionalen Anforderungen an ein Steuerungssystem nach Funktionsänderungen noch erfüllt sind. Dazu werden ein Netzwerkscan, eine modellbasierte Auswirkungsanalyse und schlussendlich eine modellbasierte Verifikation des betroffenen Teilsystems durchgeführt.

Die Struktur von TestIAS, welches in mehreren Abschnitten implementiert wurde [130]–[132], ist in Abbildung 27 illustriert. Die dort dargestellten Komponenten des TestIAS werden in diesem Kapitel erläutert.

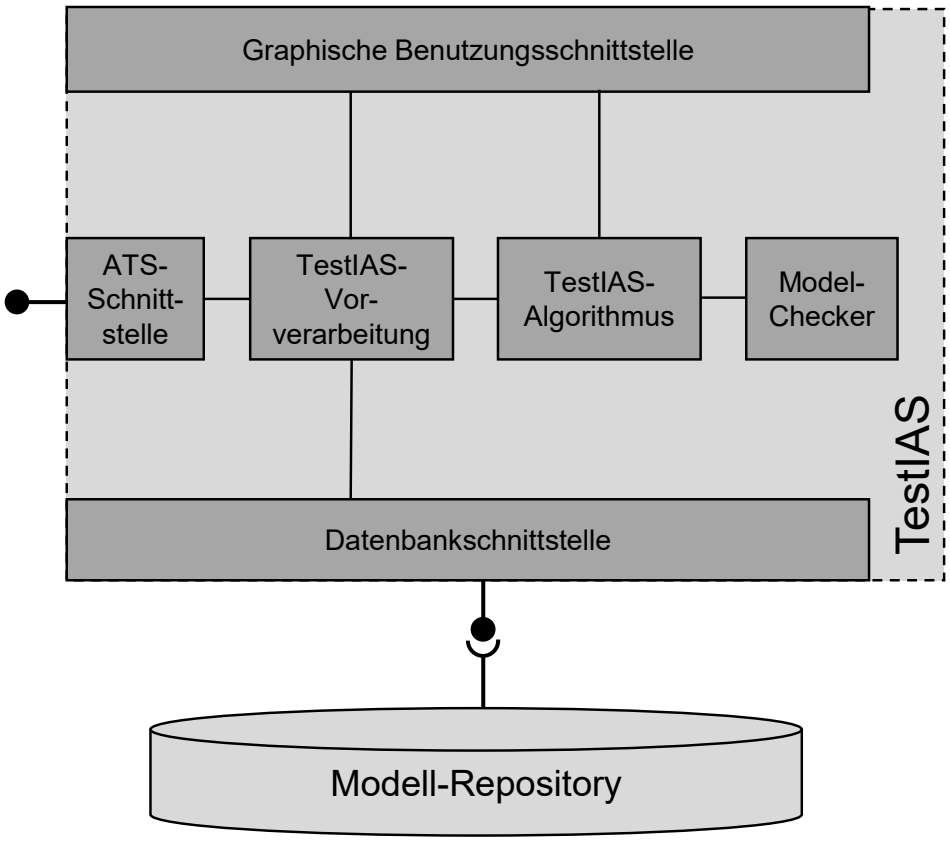


Abbildung 27: Überblick über die Struktur von TestIAS

Zu Beginn werden in Kapitel 4.1 die Informationen benannt, welche aus dem Steuerungssystem auslesbar sein müssen, um dessen Aufbau ausreichend analysieren zu können. Anschließend werden in Kapitel 4.2 der Aufbau des Modell-Repository und in Kapitel 4.3 die IT-Schnittstellen von TestIAS (ATS-Schnittstelle und Datenbankschnittstelle) zu den Informationsquellen beschrieben.

Im Anschluss daran wird in den Kapiteln 4.4 und 4.5 der funktionale Programmablauf von TestIAS erörtert. Dieser gliedert sich in die TestIAS-Vorverarbeitung, bei welcher notwendige Verhaltensmodelle bezogen und aufbereitet werden, und den TestIAS-Algorithmus, welcher das beschriebene Konzept umsetzt**.** Schließlich werden in Kapitel 4.6.1 die Ergebnisdarstellung anhand einer graphischen Benutzungsoberfäche und in Kapitel 4.6.3 der Hardwareaufbau vorgestellt.

Da TestIAS zur Absicherung eines serviceorientierten Steuerungssystems entworfen ist, soll im Folgenden nicht mehr allgemein von Komponenten, sondern spezifischer von Services gesprochen werden. Somit entsprechen die im Konzept beschriebenen „Modelle der Komponenten“ jetzt „Modellen der Services“ und „Komponenten-Anforderungen“ den „Service-Anforderungen“.

4.1 Notwendige Informationen aus einer verteilten Steuerung

In dieser Umsetzung des Konzepts müssen mithilfe eines Netzwerkscans Informationen der verteilten Steuerung bezogen werden, die deren aktuelle Konfiguration ausreichend beschreibt. Dabei ist die Information relevant, welche Services sich im Netzwerk befinden und wo sie angeordnet sind. Um die Realisierung eines Service eindeutig zu bestimmen, sind folgende Attribute notwendig:

- Servicename
- Version
- Hersteller & Typ

Wenn die Position eines Service relevant ist oder gleichartige Services in einem Steuerungssystem mehrfach vorkommen, bedarf es weiterer Informationen, um die gewollte Instanz anzusprechen. Beispielhaft dafür sind gleichartige Sensoren und Aktoren, welche an verschiedenen Orten installiert wurden oder gleichartige Maschinen, welche es zu unterscheiden gilt. Zur eindeutigen Bestimmung von Instanzen eines Service können Topologien verwendet werden. In dieser Realisierung wurde zusätzlich zu den beschriebenen Attributen folgende hierarchisch aufgebaute Topologieinformation definiert:

- Location

4.2 Notwendige Informationen aus einem Modell-Repository

Im Modell-Repository werden die Verhaltensmodelle und funktionalen Anforderungen der Services des Steuerungssystems verwaltet. Diese entstanden während der modellbasierten Entwicklung der Services und sind im Repository über folgende Attribute eindeutig beschrieben:

- Servicename
- Version
- Hersteller & Typ

Die Verhaltensmodelle der Services sind in dem standardisierten, XML-basierten Austauschformat PNML (ISO/IEC 15909) gespeichert. Ein Ausschnitt eines im PNML-Format beschriebenen Verhaltensmodells ist in Abbildung 28 dargestellt. Darin werden zuerst die Stellen und Transitionen mitsamt den Latenzzeiten definiert. Anschließend werden die Flussrelationen (Kanten) zwischen Stellen und Transitionen beschrieben. Die Petri-Netze sind so modelliert, dass sie pro Service-Typ nur einmalig erstellt und für jede Instanz des Service wiederverwendet werden können.

Ausschnitt eines Petri-Netzes im PNML-Format

Deklaration einer Stelle

```
<Place id="P2">
        <Name>MotorOn</Name>
        <Marken>0</Marken>
</Place>
```

Deklaration einer Interface-Stelle

```
<Place id="Call10401017XXYYZZ">
        <Name>CallMAoff</Name>
        <Marken>0</Marken>
</Place>
```

Deklaration einer Transition

```
<Transition id="T1">
        <Name>T1</Name>
        <earliestFiringTime>0</earliestFiringTime>
        <latestFiringTime>0</latestFiringTime>
</Transition>
```

Definition von Kanten

```
<arc id="a1" source="P1" target="T1" marking="1"/>
<arc id="a2" source="T1" target="P2" marking="1"/>
<arc id="a5" source="Call10401016XXYYZZ" target="T1" marking="1"/>
<arc id="a6" source="Call10401017XXYYZZ" target="T2" marking="1"/>
```

Abbildung 28: Ausschnitt eines im PNML-Format definierten Petri-Netzes

Die Service-Anforderungen sind in der temporalen Logik CTL spezifiziert. Für jede Service-Anforderungen ist definiert, welche Module des technischen Prozesses benötigt werden und in welchem Anfangszustand diese sich befinden. Als Service-Anforderung ist unter anderem spezifi-

ziert, dass kein Deadlock auftreten darf, eine bestimmte Stelle erreichbar sein muss oder ein Zustand nicht eintreten darf. Beispielhafte Anforderung an einen Service finden sich in Abbildung 29.

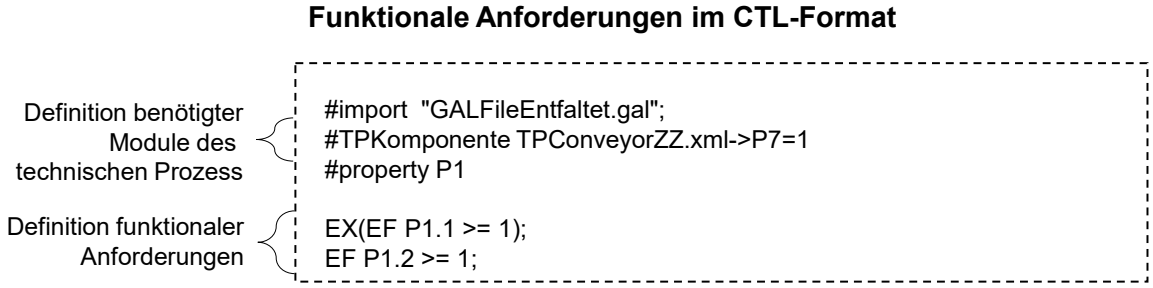


Abbildung 29: Darstellung spezifizierter Service-Anforderungen

Zu jedem Service des Steuerungssystems sind das Verhaltensmodell sowie die Anforderungen in der Modelldatenbank hinterlegt. Es wird dabei angenommen, dass in Zukunft, dank dem Trend zum Digitalen Zwilling und der modellbasierten Entwicklung, Verhaltensmodelle von Services zunehmend vom Hersteller bereitgestellt werden.

Für Services, die mehrfach in verteilten Steuerungen vorkommen können, genügt es, jeweils einmalig ein Verhaltensmodell zu erstellen, Service-Anforderungen zu definieren und diese im Modell-Repository anzulegen. Im Rahmen der TestIAS-Vorverarbeitung werden die Modelle und Anforderungen mithilfe von Topologieinformationen instanziiert.

4.3 Schnittstellen zu den Informationsquellen

Zur Ausführung des TestIAS-Algorithmus werden externe Informationsquellen benötigt. Wie in Abbildung 27 dargestellt, benötigt TestIAS Informationen aus dem verteilten Steuerungssystem sowie dem Modell-Repository.

4.3.1 Schnittstelle zum verteilten Steuerungssystem

Zur Kommunikation mit einem verteilten Steuerungssystem besitzt TestIAS einen OPC-UA-Client. Die Verwendung von OPC UA wurde gewählt, da dies als zukünftiger Industriestandard der M2M-Kommunikation gesehen wird und OPC UA einen Discovery Prozess anbietet, welcher eine einfache Integration von Komponenten in das OPC-UA-basierte Steuerungsnetzwerk ermöglicht [52]. Dieser OPC-UA-Client, welcher eine Integration von TestIAS in OPC-UA-Netzwerke ermöglicht, ist über einen Wrapper eingebunden. Ein Wrapper ist ein Stück Software, welche den OPC-UA-Client kapselt. Dies erlaubt eine einfache Austauschbarkeit der Schnittstelle für andere Protokolle von Steuerungssystemen, sodass die nachfolgend beschriebene Realisierung nicht an OPC UA gebunden ist.

Der OPC-UA-Client von TestIAS integriert sich ad hoc in das Steuerungsnetzwerk des Automatisierungssystems. Bei einer Anfrage durch die TestIAS Komponente „TestIAS-Vorverarbeitung“ liest der OPC-UA-Client das Serviceverzeichnis des Discovery-Servers aus. Die akquirierten Informationen über das Steuerungssystem werden über einen JSON-String der „TestIAS-Vorverarbeitung“ übergeben. Das Format und der Inhalt des JSON-Strings sind somit unabhängig vom Protokoll des verteilten Steuerungssystems und ermöglichen eine generische Implementierung der TestIAS-Vorverarbeitung sowie des TestIAS-Algorithmus.

4.3.2 Schnittstelle zum Modell-Repository

Der Bezug zusätzlicher Informationen über die detektierten Services ist durch eine Schnittstelle zum Modell-Repository möglich. Über die Schnittstelle können sämtliche in dem Modell-Repository verwalteten Daten bezogen werden. Daneben können neue Services über die Schnittstelle in die Datenbank eingepflegt werden. Die Verwendung einer universalen Datenbankschnittstelle erlaubt einen einheitlichen Zugriff auf Datenbanken verschiedener Hersteller. Dies ermöglicht eine einfache Erweiterung der Schnittstelle für den Zugriff auf verschiedene, zum Beispiel herstellerspezifische, Modell-Repositories.

4.4 Vorverarbeitung für den TestIAS-Algorithmus

Bei der „TestIAS-Vorverarbeitung“ werden geeignete Eingangsinformationen für den TestIAS-Algorithmus generiert. Als Ergebnis der Vorverarbeitung stehen die für das Konzept benötigten Informationen in geeigneter Form zur Verfügung. Der dreiphasige Ablauf der Vorverarbeitung, welcher in Abbildung 30 dargestellt ist, wird folgend erläutert.

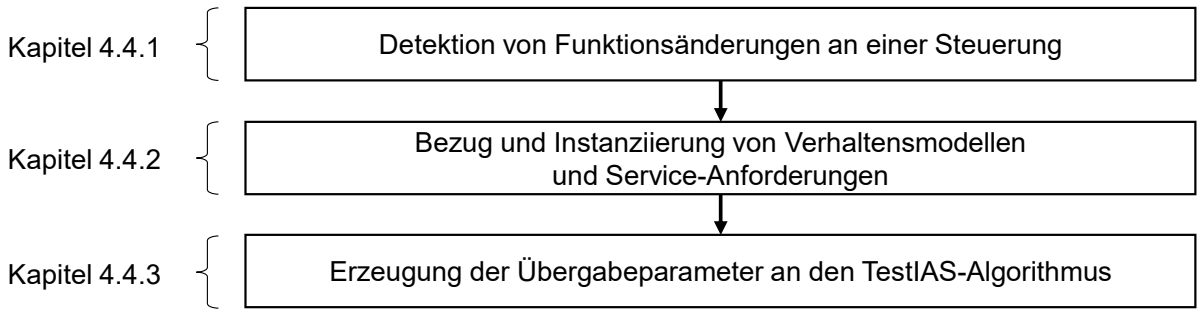


Abbildung 30: Sequenzdiagramm der TestIAS-Vorverarbeitung

4.4.1 Detektion von Funktionsänderungen an einer Steuerung

Bei Start des Programmablaufs über die graphische Benutzungsschnittstelle wird mittels der Schnittstelle zum Steuerungssystem (ATS-Schnittstelle) ein Netzwerkscan der verteilten Steuerung durchgeführt. Der durch die ATS-Schnittstelle übergebene JSON-String beinhaltet alle notwendigen Informationen, um die Services des Steuerungssystems eindeutig zu identifizieren. Anhand dieser Informationen über das Steuerungssystem wird analysiert, ob ein Service neu hinzugefügt, entfernt oder geändert wurde. Aufgrund der Versionierung von Softwareständen ist eine Änderung anhand einer veränderten Versionsnummer erkennbar. Dazu werden die durch den Netzwerkscan bezogenen Informationen mit den Informationen des vorherigen Netzwerkscans verglichen. Wird TestIAS erstmalig an die verteilte Steuerung angeschlossen, werden alle Services als „neu hinzugefügt“ klassifiziert.

4.4.2 Instanziierung von Verhaltensmodellen und Service-Anforderungen

In der darauffolgenden Phase der Vorverarbeitung werden die Verhaltensmodelle, welche als Petri-Netze vorliegen, und Service-Anforderungen aller im Netzwerkscan erkannten Services vom Modell-Repository bezogen. Gleichartige Services (Name, Hersteller & Typ und Version) beziehen sich auf das gleiche Verhaltensmodell und die gleichen Service-Anforderungen. Folglich genügt es für gleichartige Services, welche mehrfach im Steuerungssystem vorhanden sind, deren Verhaltensmodell und Service-Anforderungen einmalig zu beziehen.

Die Services werden anschließend mithilfe der Topologieinformation „Location“ instanziiert, das heißt einem konkreten Service des Steuerungssystems zugewiesen. Dieser Vorgang ist in Abbildung 31 veranschaulicht. Die generischen Modelle besitzen an ihren Schnittstellen (zu anderen Services sowie zum technischen Prozess) Platzhalter für die Location der Form „XXYYZZ“. Eine beispielhafte Umsetzung der Topologie wird in Kapitel 5.1 beschrieben. Diese Platzhalter werden bei Instanziierung durch die Location der jeweiligen Services im Steuerungsnetzwerk ersetzt. Bei den Service-Anforderungen wird analog dazu der Platzhalter durch die Information ersetzt, welches Modul des technischen Prozesses zur Verifikation benötigt wird. Durch die Individualisierung der Verhaltensmodelle und der Service-Anforderungen, werden diese somit eindeutig einem Service zuordenbar. Dadurch wird für alle im Steuerungsnetzwerk identifizierten Services ein Verhaltensmodell mit den korrespondierenden Service-Anforderungen erzeugt.

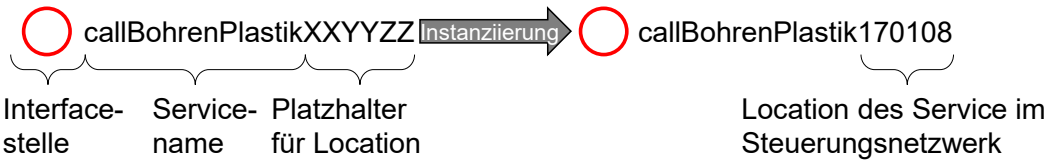


Abbildung 31: Instanziierung eines Verhaltensmodells durch Anpassung seiner Interface-Stellen

4.4.3 Übergabeparameter an den TestIAS-Algorithmus

Als Ergebnis der TestIAS-Vorverarbeitung liegen alle Eingangsdaten für den im Konzept beschrieben Algorithmus vor.

- Liste mit Services, die geändert, hinzugefügt oder entfernt wurden
- Instanziierte Verhaltensmodelle im PNML-Format sowie instanziierte Service-Anforderungen im CTL-Format

Die Informationen werden in Listen zusammengefasst und dem TestIAS-Algorithmus übergeben.

4.5 TestIAS-Algorithmus

Der TestIAS-Algorithmus setzt das in Kapitel 3 beschriebene Konzept um. So wird beschrieben, wie TestIAS Auswirkungen von Änderungen an Steuerungssystemen modellbasiert analysiert und betroffene funktionale Anforderungen verifiziert.

Die nachfolgenden Abschnitte beschränken sich somit auf realisierungsspezifische Beschreibungen und die Darstellung von Zwischenergebnissen. Bei den dargestellten Zwischenergebnissen handelt es sich um Screenshots von TestIAS bei der Verifikation des beschriebenen verteilten Steuerungssystems. Zur Erzeugung von Bilddateien aus Modellen und für die Verifikation werden Open-Source-Werkzeuge eingebunden, auf welche in Kapitel 4.6.2 verwiesen wird. Wie in Abbildung 32 skizziert folgen die anschließenden Kapitel dem Ablauf des TestIAS Algorithmus.

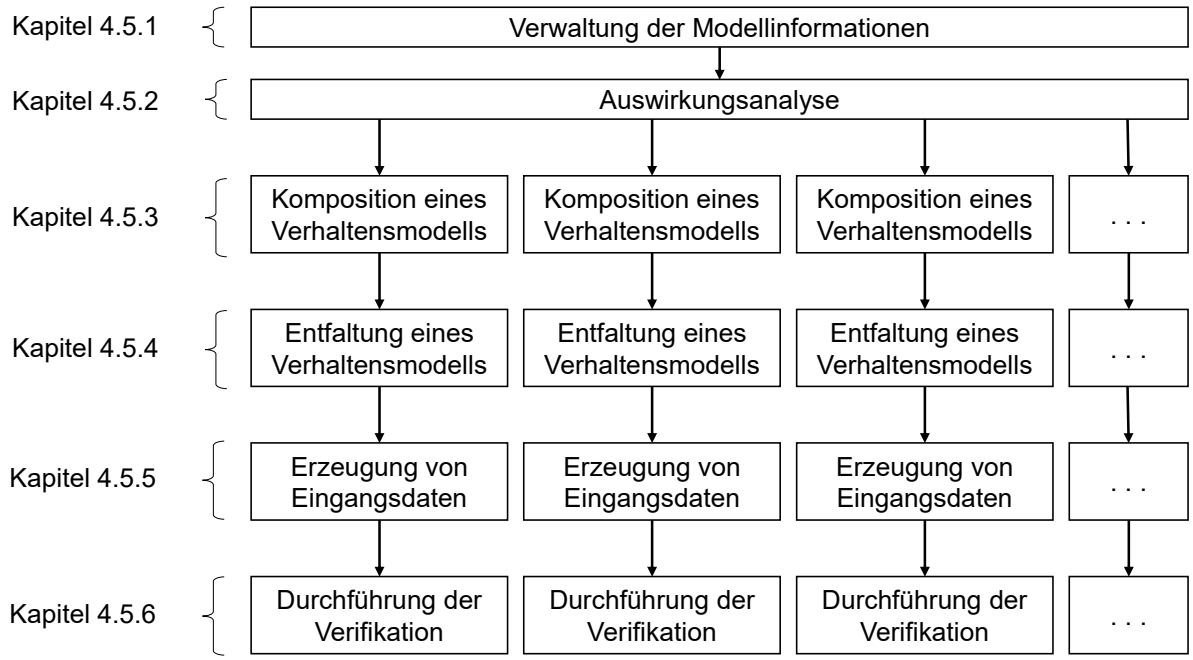


Abbildung 32: Sequenzdiagramm des TestIAS-Algorithmus

4.5.1 Verwaltung der Modellinformationen

Bei Start des TestIAS-Algorithmus werden die in der Vorverarbeitung instanziierten Petri-Netze eingelesen. Für jedes Petri-Netz wird dazu ein Objekt erzeugt, in welchem die Stellen, Transitionen und Flussrelationen des Petri-Netzes in Listen abgespeichert werden. Die Flussrelationen beinhalten Informationen über die Vorgänger- und Nachfolger-Stelle oder -Transition und das Kantengewicht. Die Transitionen beinhalten Informationen über die Latenzzeit. Somit entsteht für jeden Service des verteilten Steuerungssystems ein Objekt. Zur graphischen Darstellung der Petri-Netze wird aus jedem Objekt ein DOT-File generiert, aus welchem mit Graphviz eine Bilddatei erzeugt wird (siehe Abbildung 33).

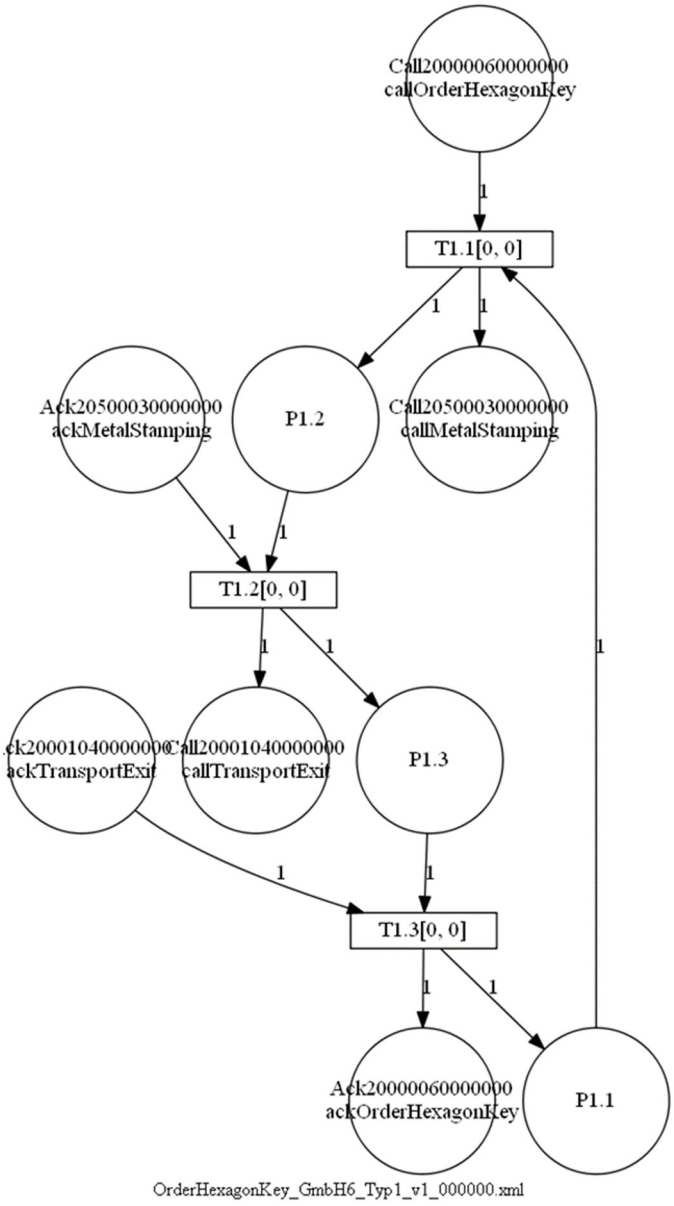


Abbildung 33: Darstellung der generierten Bilddatei des Service "Hexagon Key". Dieser Service ruft sequenziell die Services „Metal Stamping“ und „Transport Exit“ auf.

4.5.2 Durchführung der Auswirkungsanalyse

Zur Identifikation der Service-Anforderungen, welche erneut abgesichert werden müssen, wird ausgehend von den Services, die neu hinzugefügt, geändert oder entfernt wurden, eine Auswirkungsanalyse durchgeführt. Zur Analyse der Abhängigkeiten wird zunächst der Abhängigkeitsgraph in Form eines Blockdefinitionsdiagramms aufgebaut. Dazu werden die Stellen-Listen aller Objekte nach call-Stellen durchsucht. Wird eine call-Stelle erkannt, wird in der korrespondierenden Flussrelationen-Liste untersucht, ob es sich um eine Input- oder eine Output-Stelle handelt. Eine Input-Stelle stellt einen angebotenen Service dar und bildet somit das Ende eines Abhängigkeitspfeils. Eine Output-Stelle stellt den Aufruf eines Service dar und bildet somit den Ausgangspunkt eines Abhängigkeitspfeils. Alle Input- und Output-Stellen des gleichen Serviceaufrufs werden miteinander über Pfeile verknüpft.

Ausgehend von den geänderten Services werden anschließend, anhand des generierten Blockdefinitionsdiagramms, die Services identifiziert, deren Anforderungen erneut abgesichert werden müssen.

Das Blockdefinitionsdiagramm wird bei TestIAS textuell im DOT-Format beschrieben und in eine Bilddatei umgewandelt. Ein von TestIAS generiertes Blockdefinitionsdiagramm ist in Abbildung 34 dargestellt. Im oberen Teil der Abbildung sind die Abhängigkeiten aller Services des in Kapitel 5 beschriebenen Steuerungssystems aufgezeigt. Im unteren Teil der Abbildung ist ein Teilbereich vergrößert hervorgehoben. Services, deren funktionale Anforderungen erneut abgesichert werden müssen, werden in der Bilddatei hellblau hervorgehoben. Ferner werden aus dem Steuerungsnetzwerk entfernte Services rot dargestellt. Eine Detailansicht des gesamten Blockdefinitionsdiagramms befindet sich im Anhang.

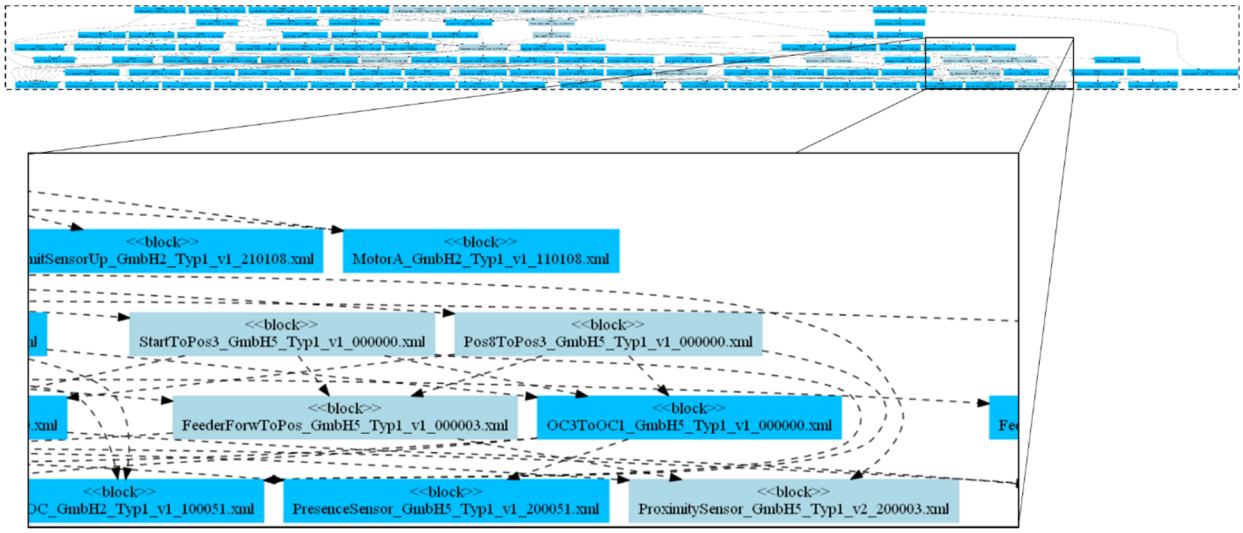


Abbildung 34: Screenshot eines generierten Blockdefinitionsdiagramms

Als Ergebnis der Auswirkungsanalyse liegt eine Liste mit den Services vor, deren Anforderungen erneut abgesichert werden müssen. Für diese Service-Anforderungen wird im nächsten Schritt ein dazu passendes Petri-Netz komponiert.

4.5.3 Komposition der Verhaltensmodelle

Für jeden Service, dessen funktionale Anforderungen erneut abgesichert werden müssen, wird im folgenden Schritt ein Teilsystem komponiert, das alle Services enthält, von denen der jeweilige Service abhängig ist. Dazu wird für jeden abzusichernden Service ein neues Objekt erzeugt, welches die Listen (Stellen, Transitionen und Flussrelationen) der zu komponierenden Services nach den Kompositionsregeln aus Kapitel 3.4.2 zusammenführt.

Die Stellen und Transitionen der Verhaltensmodelle der einzelnen Services besitzen die gleiche Namenskonvention (P1, P2, .../ T1, T2, …). Um auch nach einer Komposition die Stellen den Teilnetzen zuordnen zu können, werden diese vorab um eine Ziffer erweitert, welche spezifisch für den jeweiligen Service ist. Eine Ausnahme bilden dabei die Interface-Stellen und Schnittstellen zum technischen Prozess, deren Bezeichnung unverändert bleibt. Die komponierten Petri-Netze werden ebenfalls in eine Bilddatei umgewandelt. Ein komponiertes Netz ist exemplarisch in Abbildung 35 zu sehen. Das komponierte Petri-Netz des Gesamtsystems ist im Anhang detailliert dargestellt. Die Verhaltensmodelle der einzelnen Services werden verschiedenfarbig angezeigt.

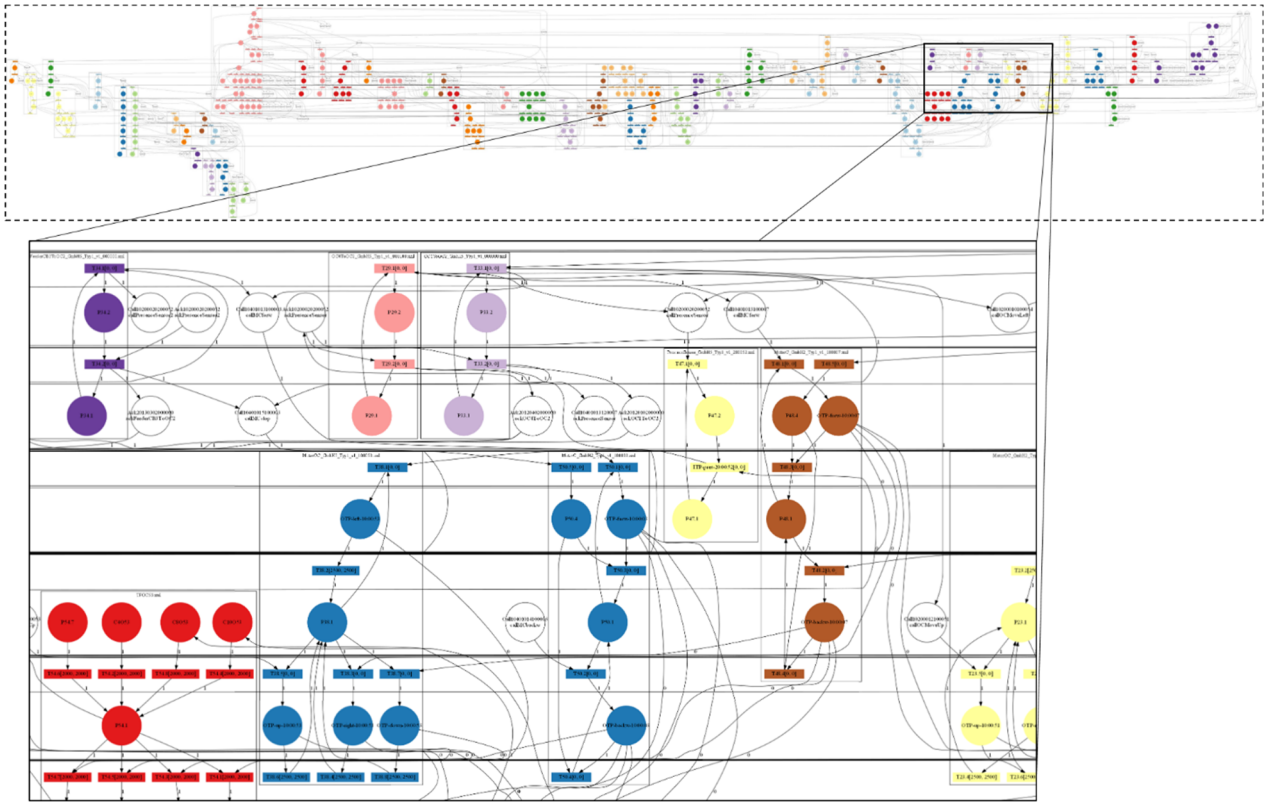


Abbildung 35: Screenshot eines komponierten Petri-Netzes

4.5.4 Farbige Erweiterung und Entfaltung der komponierten Verhaltensmodelle

Aufgrund der in Kapitel 3.4.3 beschriebenen Problematik des Mehrfachzugriffs beschreibt das komponierte Petri-Netz das Verhalten des Teilsystems noch nicht ausreichend. Zur Vermeidung des Abziehens eines Tokens durch einen unbeteiligten Service wurde in Kapitel 3.4.3 ein Konzept dazu entworfen, wie die Petri-Netze der betroffenen Services mithilfe farbiger Tokens erweitert werden können. Dies garantiert, dass nach Durchführung der Funktionalität des Service der Token zum aufrufendem Services zurücktransportiert wird. Dies wurde in TestIAS wie folgt umgesetzt.

Anhand des Blockdefinitionsdiagramms wird erkannt, welche Services von verschiedenen Services aufgerufen werden können. Wird eine Bestätigungs-Nachricht (ack) an die aufrufenden Services zurückgesendet, müssen die aufgerufenen Services zu farbigen Petri-Netzen erweitert werden. Um das zu erkennen, wird die Stellen-Liste des mehrfach aufrufbaren Petri-Netzes nach der Existenz der entsprechenden ack-Stelle durchsucht.

Zur Erkennung des farbig zu erweiternden Pfads wird von der Call-Stelle ausgehend eine Breitensuche durchgeführt. Die Kanten der Pfade, welche die ack-Stelle besitzen, werden bis zur ack-Stelle farbig erweitert.

Da das verwendete Verifikationswerkzeug nicht mit farbigen Petri-Netzen kompatibel ist, werden diese durch TestIAS zusätzlich entfaltet. Die Entfaltung wandelt das farbige Petri-Netz in ein konventionelles Stellen/Transitions-Netz um, welches dasselbe Verhalten beschreibt. Durch die Entfaltung erzeugt der in TestIAS implementierte Algorithmus parallele Pfade von der Call-Stelle bis zur Ack-Stelle. In Abbildung 36 ist die farbige Erweiterung und Entfaltung beispielhaft bei zwei aufrufenden Komponenten dargestellt. Die farbige Erweiterung geschieht gemäß des in Kapitel 3.4.3 beschriebenen Verfahrens. Dadurch wird bei Schalten der Transition *T1* von Service A oder Service B ein Token der Farbe ID=A respektive ID=B bei erzeugt. Wie anhand der Markierung „ID“ an den Kanten deutlich wird, wird dieses Token bis zur Stelle *ackServiceX* weitergegeben. Diese Stelle bildet eine Alternative. Dabei muss das Token zurück zur aufrufenden Komponente gelangen. Dies wird gewährleistet, indem die Transitionen *T2* der Services A und B über einen Guard ([ID=A] und [ID=B]) verfügen, der ein Schalten nur zulässt, wenn das Token die passende Farbe besitzt.

Um wieder ein einfaches Stellen/Transitionen-Netz zu erhalten wird das Petri-Netz anschließend entfaltet. Die Anzahl der parallelen Pfade, die bei der Entfaltung generiert werden ist davon abhängig, von wie vielen Services der Service aufgerufen werden kann. Bei drei aufrufenden Services entsteht analog dazu ein dritter paralleler Pfad. Petri-Netze werden somit durch Entfaltung für mehr IT-Werkzeuge interpretierbar, sind aber schwerer für Menschen nachvollziehbar.

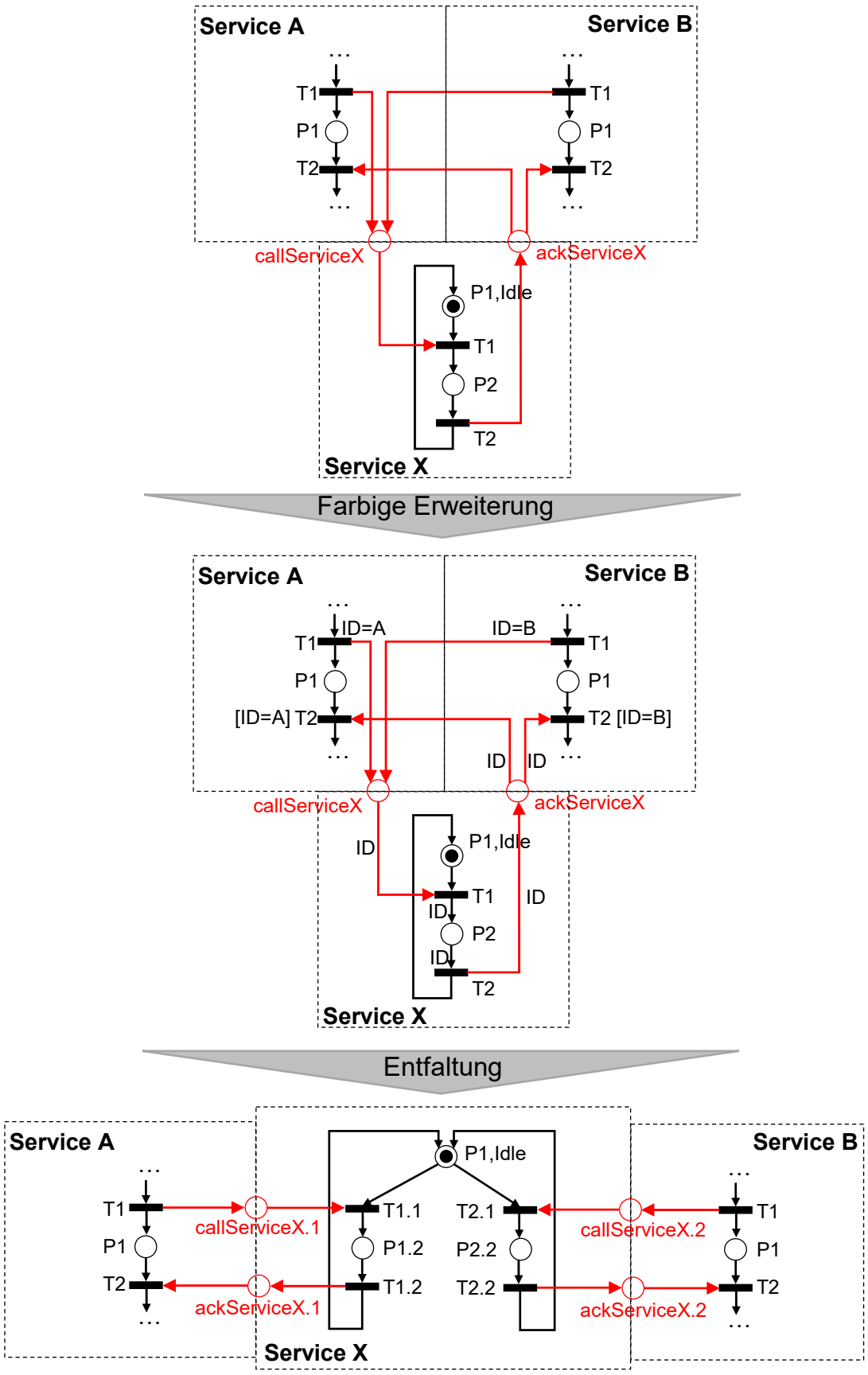


Abbildung 36: Farbige Erweiterung und Entfaltung mehrfach aufgerufener Komponenten

4.5.5 Erzeugung der Eingangsdaten für die Verifikation

Zur Verifikation des komponierten Verhaltensmodells eines Steuerungsnetzwerks sind folgende Schritte notwendig:

- Anbindung des technischen Prozesses
- Erzeugung eines Initiierungs-Tokens
- Umwandlung in ein für Verifikationswerkzeuge kompatibles Datenformat
- Bezug der Service-Anforderungen

Aus den Service-Anforderungen der abzusichernden Komponente wird ausgelesen, welche Module des technischen Prozesses zur Verifikation benötigt werden. Die Verhaltensmodelle dieser Module des technischen Prozesses werden mit dem erzeugten Verhaltensmodell der Steuerung komponiert.

Um den zu verifizierenden Service in den Anfangszustand zu versetzen, wird in seiner call-Stelle ein zusätzlicher Token erzeugt. Dies erweitert die Startmarkierung des Petri-Netzes zu dem Zustand, welchen das Teilsystem bei Serviceaufruf aufweist, und initiiert somit die Durchführung der Funktionalität.

Nach Anpassung der Startmarkierung des komponierten Petri-Netzes wird das Petri-Netz in das Datenformat des verwendeten Model Checker transformiert. Die erzeugte Datei mit den zugehörigen Service-Anforderungen wird anschließend dem Model Checker ITS-Tool übergeben.

4.5.6 Durchführung der Verifikation

Nach Generierung der Eingangsdaten werden die Verifikationsprozesse ausgeführt. Der Verifikationsprozess der Anforderungen eines Service kann mit dem verwendeten Model Checker auf dem zur Verfügung stehenden PC mehrere Stunden dauern. Um möglichst schnell den Verifikationsprozess abzuschließen und dadurch einen praxistauglichen Einsatz gewährleisten zu können, werden möglichst viele Verifikationsprozesse parallel ausgeführt. TestIAS führt fünf Verifikationsprozesse parallel aus. Dieser anpassbare Parameter wurde so gewählt, damit der PC eine maximale Prozessor-Auslastung bei geringem Arbeitsspeicherbedarf hat. Versuche zeigten, dass bei höheren Werten paralleler Verifikationsprozesse sich die Prozesse gegenseitig ausbremsen und den Arbeitsspeicher belasten. Ist ein Verifikationsprozess abgeschlossen, startet der nächste noch wartende Verifikationsprozess, bis alle durchgeführt sind. Die Dauer der einzelnen Verifikationsprozesse und die Gesamtdauer der Verifikation werden gemessen und in Text-Dateien abgelegt. Die benötigte Dauer der Verifikationsprozesse wird im Rahmen der Evaluierung in Kapitel 6.3.1 betrachtet.

4.6 Implementierung des Demonstrators

Die beschriebene Realisierung des Konzepts wurde in Form des Demonstrators TestIAS implementiert. Implementierungsspezifische Informationen, über die realisierte Benutzungsschnittstelle, die verwendete Programmiersprachen und den Hardwareaufbau werden im Folgenden gegeben.

4.6.1 Graphische Benutzungsschnittstelle

Zur Gewährleistung der Benutzbarkeit von TestIAS ist die Benutzungsschnittstelle essentiell. Sie dient dazu, den Absicherungsprozess zu starten, das Verifikationsergebnis zu analysieren und das Modell-Repository zu pflegen. Ein Screenshot der entworfenen Benutzungsschnittstelle findet sich in Abbildung 37.

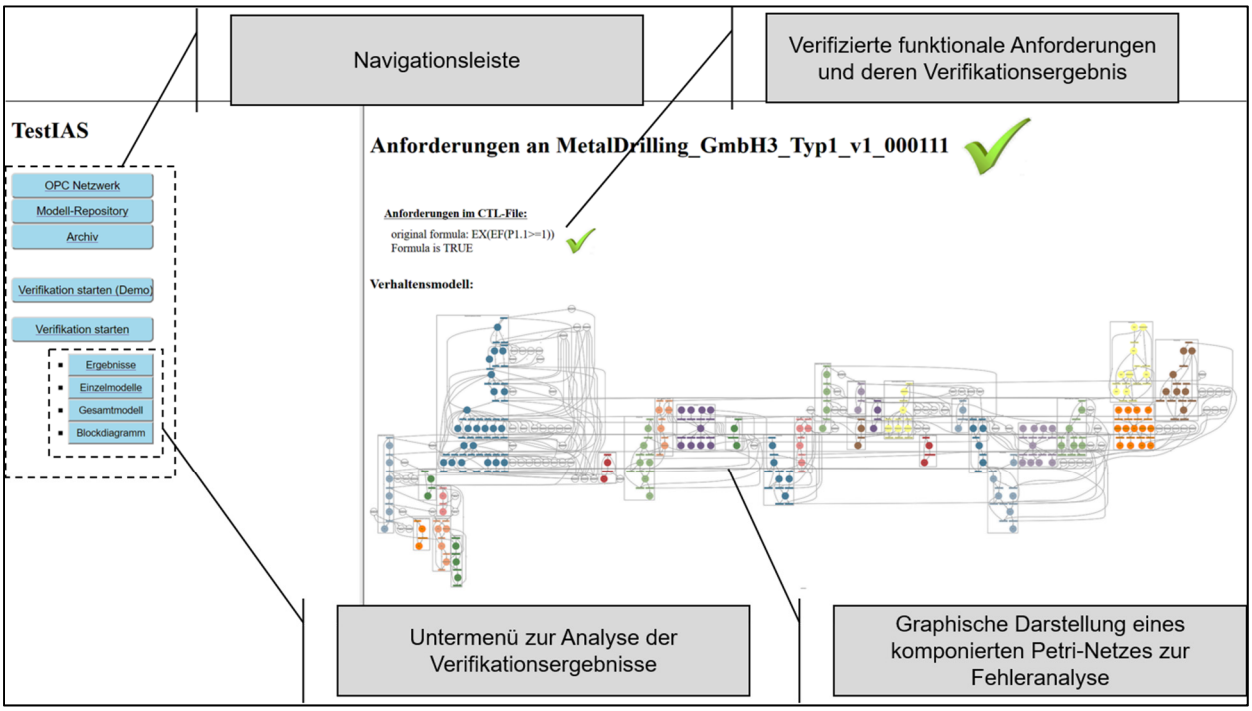


Abbildung 37: Graphische Benutzungsschnittstelle von TestIAS

4.6.1.1 Start der Verifikation

Bei Betätigen von „Verifikation Starten“ wird der TestIAS-Algorithmus durchlaufen und anschließend das Verifikationsergebnis graphisch dargestellt. Da der Verifikationsprozess mehrere Stunden dauern kann (siehe Kapitel 6.3.2.1) wurde auch eine Demo-Version des TestIAS-Algorithmus erstellt, welcher einen Netzwerkscan durchführt und das damit korrespondierende Verifikationsergebnis aus einem Archiv-Ordner lädt.

4.6.1.2 Darstellung der Verifikationsergebnisse

Nach Abschluss des Verifikationsprozesses wird über einen großen grünen Haken beziehungsweise ein großes rotes Kreuz symbolisiert, ob die Verifikation aller Anforderungen erfolgreich war. Bei fehlgeschlagener Verifikation kann im Unterpunkt „Ergebnisse“ für jeden Service, der verifiziert wurde, nachvollzogen werden, ob dessen Service-Anforderungen erfüllt sind.

Die Detailansicht für den Service „MetalDrilling“ der Firma „GmbH3“ vom Typ „Typ1“ an der Location „000111“ ist in Abbildung 37 dargestellt. Ob die jeweilige, in CTL definierte, Komponentenanforderung erfüllt ist, wird mit einem Haken symbolisiert. Darunter ist das komponierte Verhaltensmodell des Teilsystems der Steuerung abgebildet, welches zur Fehlerfindung genutzt werden kann. Darüber hinaus ist über den Unterpunkt Blockdiagramm nachvollziehbar, von welchen Services ein fehlgeschlagener Service abhängt. Dazu ist die generierte Bilddatei des Blockdefinitionsdiagramms (siehe Abbildung 34) eingebettet. Zur Überprüfung des Verhaltens eines isolierten Service lassen sich dessen Verhaltensmodelle in dem Unterpunkt „Einzelmodell“ anzeigen. Hierbei sind die generierten Bilddateien der Services aus Abbildung 33 eingebunden. Im Unterpunkt „Gesamtmodell“ wird das komponierte Verhaltensmodell des gesamten Steuerungssystems dargestellt. Das Gesamtmodell des in Kapitel 5 beschriebenen verteilten Automatisierungssystems ist im Anhang detailliert dargestellt. Dies dient in erster Linie der Visualisierung der hohen Komplexität der Steuerungssoftware.

4.6.1.3 Administration des Modell-Repository

Die Administrationsoberfläche, die in Abbildung 38 dargestellt ist, erlaubt eine komfortable Pflege des Modell-Repository. Zeilenweise sind die Einträge des Modell-Repository dargestellt. Wie in Kapitel 4.2 erläutert ist jeder Service eindeutig über den Servicenamen, den Typ und die Versionsnummer identifizierbar. Zu jedem Service ist dazu das korrespondierende Petri-Netz im PNML-Format und die funktionalen Anforderungen im CTL-Format abgelegt. Diese Einträge lassen sich über die Webapplikation einsehen, hinzufügen, entfernen und ändern.

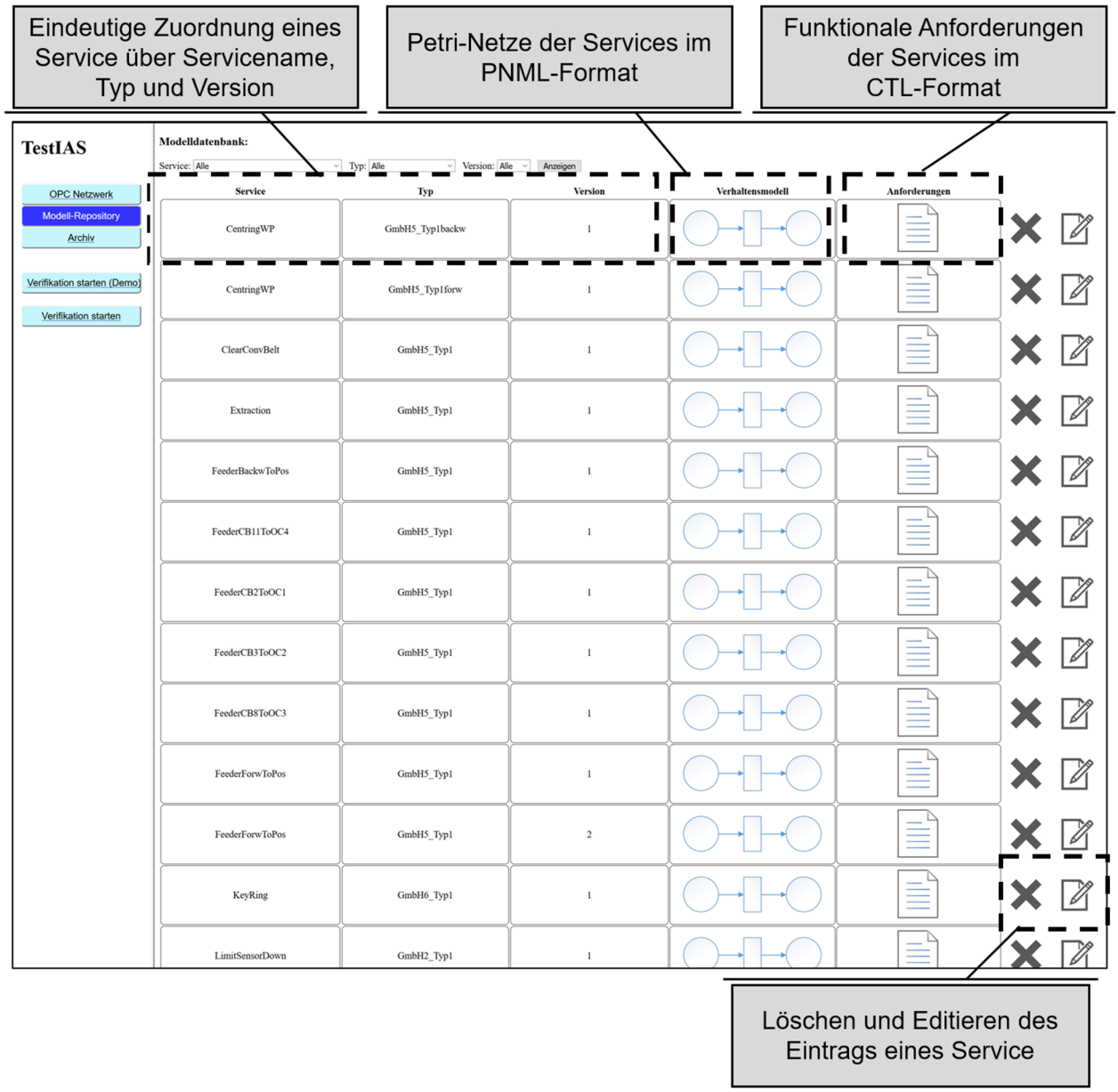


Abbildung 38: Administrationsoberfläche des Modell-Repository

4.6.2 Verwendete Programmiersprachen und Werkzeuge

Die beschriebene Realisierung des Konzepts wurde mithilfe einiger Programmiersprachen und bestehender Software-Werkzeuge implementiert, auf welche im Folgenden eingegangen wird.

Die TestIAS-Vorverarbeitung ist in der Programmiersprache Java geschrieben. Zum Parsen der Petri-Netze, welche in dem auf XML basierende Format PNML vorliegen, wird der DOM-Parser (Document Object Model-Parser) verwendet.

Die ATS-Schnittstelle verwendet zum Zugriff auf ein OPC-UA-Netzwerk einen java-basierten OPC-UA-Client des Herstellers Prosys [133]. Dieser ermöglicht das Abrufen des Service eines Discovery-Servers, welcher eine Liste der angebotenen Funktionalitäten (Services) innerhalb des OPC-UA-Netzwerks zurückgibt.

Die Schnittstelle der TestIAS-Vorverarbeitung zum Modell Repository wurde mithilfe des Java Database Connectors (JDBC) realisiert. Die JDBC-Schnittstelle zeichnet sich dadurch aus, dass sie einen einheitlichen Zugriff auf verschiedene Datenbanksysteme ermöglicht. So existiert die Möglichkeit, weitere Datenbanken einfach zu integrieren.

Zur Verwaltung des Modell-Repository wird eine MySQL Datenbank verwendet. Diese ist auf einem Apache-Server eingerichtet.

Der TestIAS-Algorithmus ist ebenfalls in Java geschrieben. Zur automatisierten Erzeugung der Bilddateien (.png) aus den generierten Modellen ist das open-source Software-Werkzeug Graphviz eingebunden [134]. Zur Erzeugung der Bilddateien mit Graphviz wurde ein Algorithmus geschrieben, der Petri-Netze und das Blockdefinitionsdiagramm in das Format .DOT konvertiert.

Als Verifikationsumgebung wird der open-source Model Checker ITS-Tool verwendet. Dieser ermöglicht die Verifikation von Petri-Netzen gegen in CTL-spezifizierte funktionale Anforderungen unter der Berücksichtigung zeitlicher Aspekte. Für die Nutzung des ITS-Tool wurde ein Algorithmus implementiert, der die komponierten Petri-Netze in das Format .GAL konvertiert. Die Verwendung des Model Checker ist über eine graphische Benutzungsschnittstelle oder eine Kommandozeile möglich. In dieser Implementierung ist das ITS-Tool über die Kommandozeile angebunden. Das ITS-Tool liegt als ausführbare Datei (.exe) vor.

Die graphische Benutzungsschnittstelle ist als Webapplikation realisiert, welche auf einem TomCat-Server der Version 7.0 integriert ist. Aufgrund der einfachen Integrierbarkeit von Java-Code in HTML-Seiten wurde die Webapplikation mithilfe der Programmiersprache Java Server Pages (JSP) erstellt.

4.6.3 Hardwareaufbau TestIAS

TestIAS ist in Abbildung 39 dargestellt. In ein Messgerätegehäuse des Typs „Europa“ der Firma OKW (Odenwälder Kunststoffwerke Gehäusesystem GmbH) wurde ein 10,1 Zoll Touch-Display der Firma Joy-IT [135] und eine Ethernet-Buchse eingelassen, über welche sich TestIAS in ein Steuerungsnetzwerk integrieren lässt. Neben dem Ethernetanschluss benötigt der Hardwareaufbau nur noch eine Spannungsversorgung über das 230V-Stromnetz.

Im Gehäuse befindet sich ein Mini-PC mit folgender Spezifikation:

- Modell: Fujitsu Esprimo Q957
- Betriebssystem: Microsoft Windows 10 Home
- Prozessor: Intel(R) Core(TM) i7-7700T, 2.90 GHz, 4 physische Kerne
- Arbeitsspeicher: 32 GB RAM

Die beschriebene Implementierung von TestIAS lässt sich über das Touch-Display bedienen und über den Mini-PC ausführen.

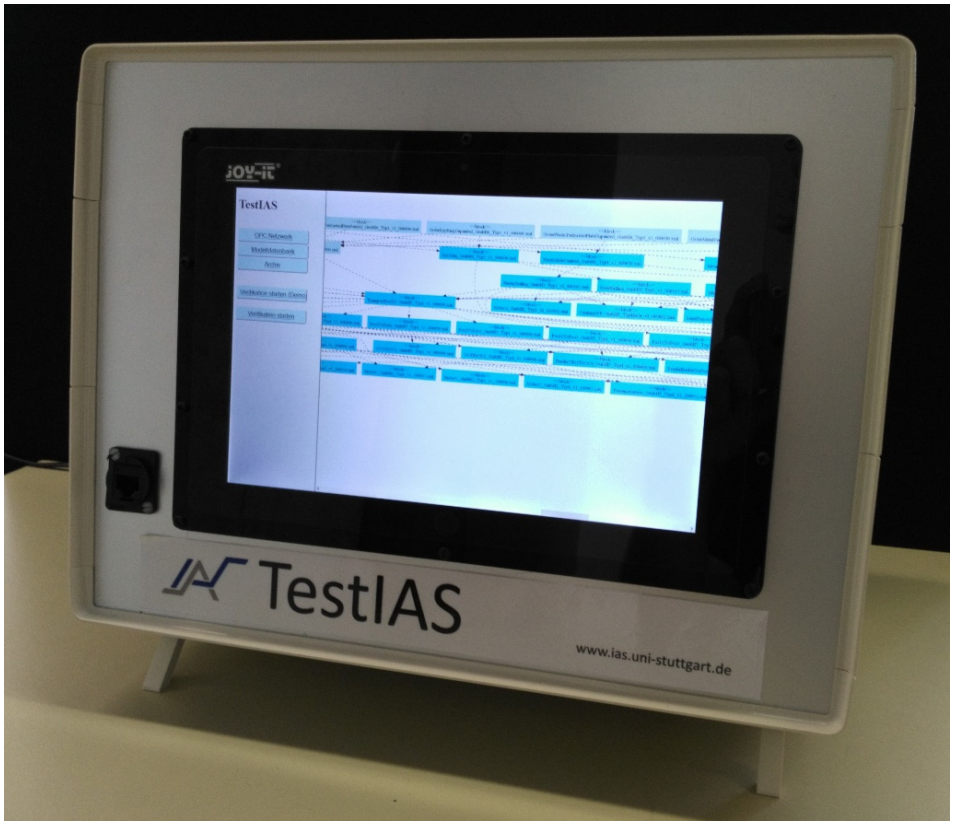


Abbildung 39: Hardware-Aufbau TestIAS

5 Verteiltes Automatisierungssystem als Testobjekt

Um die Funktionsweise von TestIAS zu evaluieren, bedarf es eines verteilten, änderbaren Steuerungssystems. Zur Darlegung der Nutzbarkeit für zukünftige Automatisierungssysteme wurde beschlossen, dass dies einigen zukunftsweisenden Eigenschaften genügen soll (siehe Kapitel 2.1.2). Das beinhaltet eine Steuerung mit verteilter Struktur, Fähigkeit zur Ad-hoc-Vernetzung, Software-Änderbarkeit, Redundanz und Wiederverwendbarkeit von Komponenten. Da solche Systeme kommerziell nicht zur Verfügung stehen, wurde entschieden, ein Automatisierungssystem zu entwerfen und aufzubauen, welches diesen Eigenschaften zukünftiger Steuerungssysteme genügt. Der Aufbau des Automatisierungssystems wird nachfolgend erläutert.

In Abbildung 40 ist die Struktur des Automatisierungssystems illustriert. Sie besteht aus einem technischen Prozess, der durch ein verteiltes Steuerungssystem beeinflusst wird. Das Steuerungssystem lässt sich über eine graphische Benutzungsschnittstelle administrieren. Die Benutzungsschnittstelle beinhaltet zum einen die Vergabe von Produktionsaufträgen, zum anderen die Möglichkeit zur Initiierung von Software-Änderungen am Steuerungssystem. Eine Schnittstelle des Steuerungssystems erlaubt die Integration von TestIAS. Darüber lassen sich Änderungen der Steuerungssoftware detektieren und mithilfe dieser Informationen eine Absicherung des Automatisierungssystems realisieren.

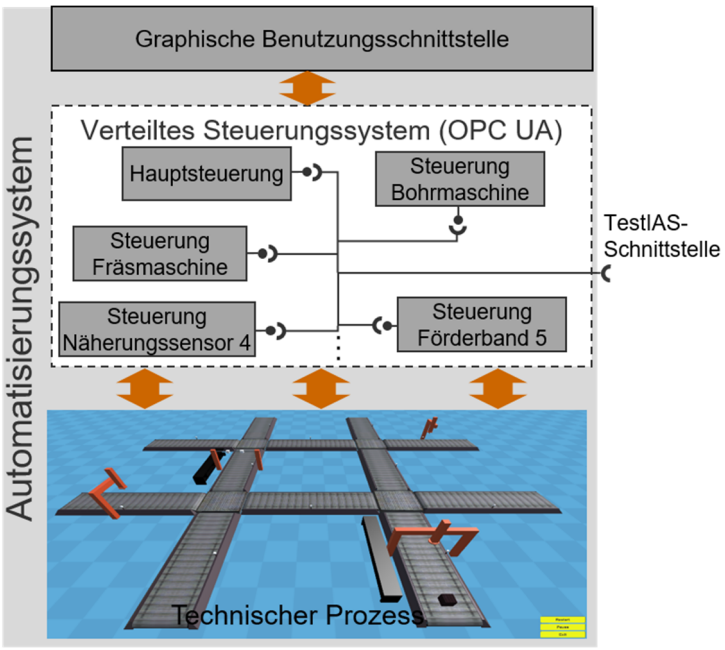


Abbildung 40: Überblick über das realisierte Automatisierungssystem

Zur Realisierung des Automatisierungssystems wurden, im Rahmen zahlreicher kleinerer Arbeiten [136]–[139], vielversprechende Technologien verwendet und innovative Mechanismen implementiert, welche im Folgenden erläutert werden.

5.1 Technischer Prozess

Der in Abbildung 41 dargestellte technische Prozess beschreibt eine diskrete Fertigung. Dieser verfügt über vier multifunktionale Bearbeitungsstationen. Die teilweise redundanten Fähigkeiten der Bearbeitungsstationen sind in Tabelle 6 aufgelistet.

Tabelle 6: Darstellung der elementaren Fähigkeiten der Bearbeitungsstationen

Fähigkeit / Bearbeitungsstation	Plastik bohren	Metall bohren	Plastik fräsen	Metall fräsen	Metall stanzen	Lackieren	Absaugen
Bohr- / Stanzmaschine 1	X	X			X		
Bohr- / Stanzmaschine 2	X	X			X		
Bohr-Fräsmaschine	X		X	X			X
Lackieranlage						X	X

Die Intralogistik erfolgt in diesem Beispiel durch ein flexibles Förderbandsystem, welches aus zwölf bidirektionalen Förderbändern und vier omnidirektionalen Förderelementen besteht. Diese ermöglichen es, gemäß des One-Piece-Flows, ein Werkstück in beliebiger Reihenfolge zu den Bearbeitungsstationen zu transportieren. Der technische Prozess ist modular aufgebaut. So stellt jede Bearbeitungsstation und jedes Förderelement ein isoliertes Modul dar. Er besitzt insgesamt 22 Sensoren und 26 Aktoren.

Aufgrund der daraus resultierenden hohen mechanischen Komplexität des technischen Prozesses wurde entschieden, diesen als Simulation zu realisieren. Die Simulation wurde über eine 3D-Grafik-Engine implementiert, welche eine realitätsnahe visuelle Darstellung ermöglicht. Dabei stellen die Sensoren und Aktoren isolierte Einheiten dar, welche einzeln über Sensor- und Aktorsignale ausgelesen und angesteuert werden. Diese Signale sind in Netzwerknachrichten eingebettet, über welche die Simulation mit den korrespondierenden Komponenten des Steuerungssystems kommuniziert.

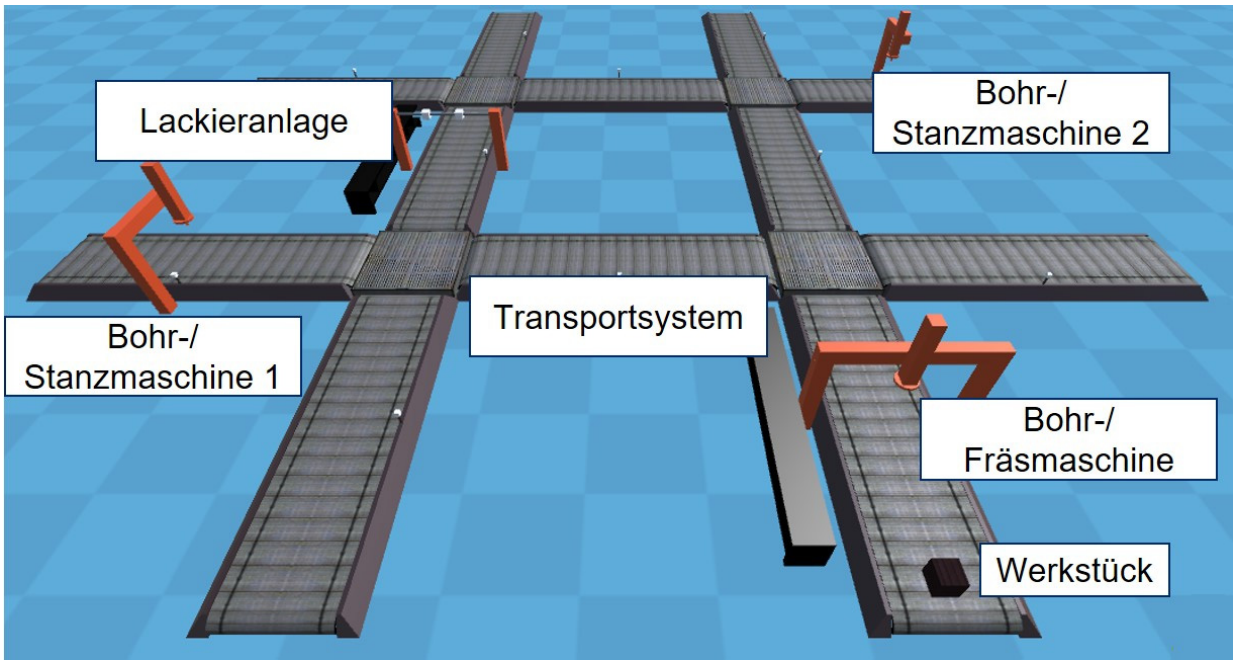


Abbildung 41: Technischer Prozess des realisierten Automatisierungssystems

Der technische Prozess ist über eine Topologie hierarchisch strukturiert. Die Topologie ermöglicht es, jeder Komponente eine logische Position zuzuweisen. Daran lässt sich beispielsweise erkennen, wo ein Sensor installiert ist oder welche Bearbeitungsstation eine Steuerungskomponente beeinflusst. Dazu wird der technische Prozess hierarchisch in Subsysteme untergliedert. Die Beschreibung einer logischen Position mittels der Topologie ist in Abbildung 42 dargestellt. Die drei Hierarchieebenen, in die der technische Prozess untergliedert ist, sind durch jeweils zwei Ziffern beschrieben und werden durch Doppelpunkte getrennt. Die unterste Hierarchieebene beschreibt die Position eines Bausteins, zum Beispiel eines Sensors innerhalb einer Bearbeitungsstation. Die mittlere Hierarchieebene beschreibt die Position von Bearbeitungsstationen und Sensoren an einem Förderband und die oberste Hierarchieebene beschreibt die Position des Förderbands in der Anlage.

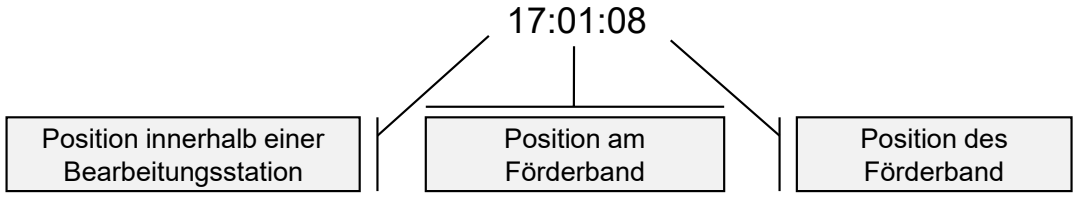


Abbildung 42: Aufbau der Topologie zur hierarchischen Beschreibung der Position einer Komponente innerhalb des technischen Prozesses

5.2 Verteiltes Steuerungssystem

Das verteilte Steuerungssystem, welches den beschriebenen technischen Prozess steuert, ist hardwareseitig auf sechs Einplatinenrechnern sowie einem PC verteilt (siehe Abbildung 43). Als Einplatinenrechner werden „Raspberry Pi 3 Model B“ der Raspberry Pi Foundation verwendet. Diese sind über Ethernet mit einem Router verbunden. Bei einem Einplatinenrechner findet die Kommunikation, statt über WLAN, kabelgebunden statt. Dies erlaubt die eingängige Darstellung einer Ad-hoc-Vernetzung durch Ein- und Ausstecken des Ethernet-Kabels.

Vier Einplatinenrechner steuern jeweils eine Bearbeitungsstation. Der fünfte Einplatinenrechner steuert das Förderbandsystem und der sechste Einplatinenrechner koordiniert übergeordnete Prozessschritte und Produktionsaufträge. Dieser Einplatinenrechner verfügt zusätzlich über ein Touch-Display, über welches Funktionsänderungen initiiert werden können.

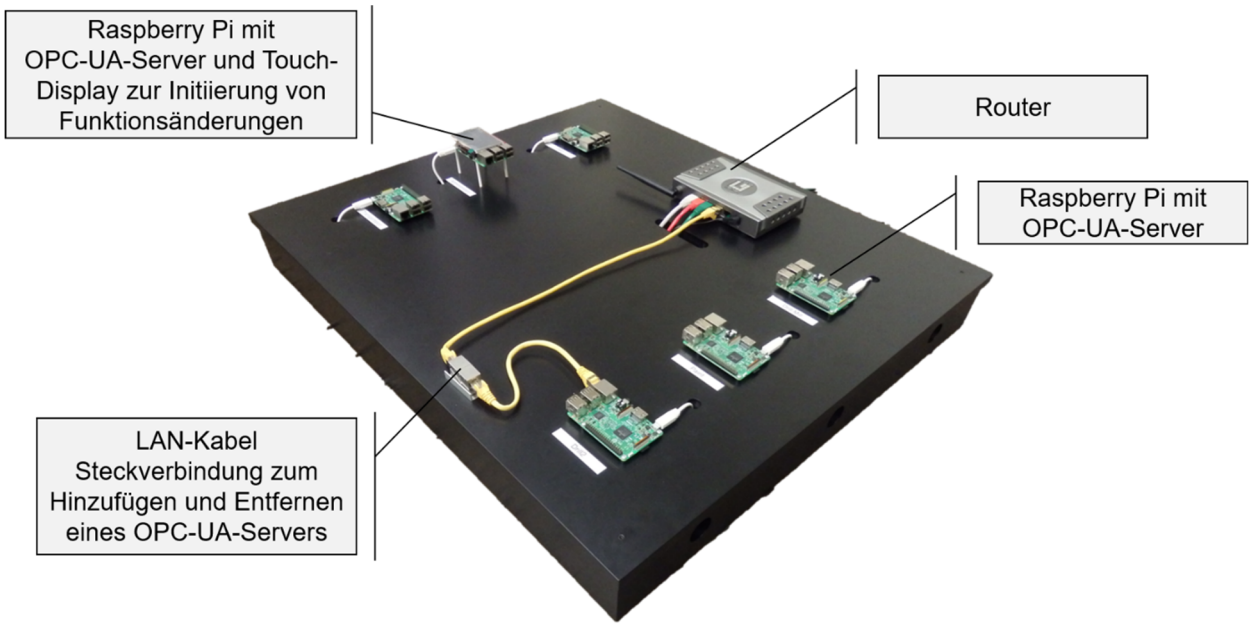


Abbildung 43: Aufbau des verteilten Steuerungssystems zur Koordination des technischen Prozesses

Die Kommunikation zwischen den Steuerungen erfolgt über das serviceorientierte M2M-Kommunikationsprotokoll OPC UA. Entsprechend Abbildung 44 verfügt jeder Einplatinenrechner über einen Java-basierten OPC-UA-Server. Jeder OPC-UA-Server beinhaltet auch einen OPC-UA-Client, um einen bidirektionalen Kommunikationsaufbau zu ermöglichen (siehe Abbildung 4). Wie in Abbildung 44 angedeutet, ermöglicht dies eine direkte Kommunikation zwischen den OPC-UA-Servern bei der Koordinierung eines Produktionsauftrags. Auf dem PC, der sich auch im Steuerungsnetzwerk befindet, ist ein Discovery-Server implementiert, welcher eine allgemein bekannte Netzwerkadresse besitzt und Informationen über im Netzwerk vorhandene OPC-UA-Server in einer Datenbank verwaltet.

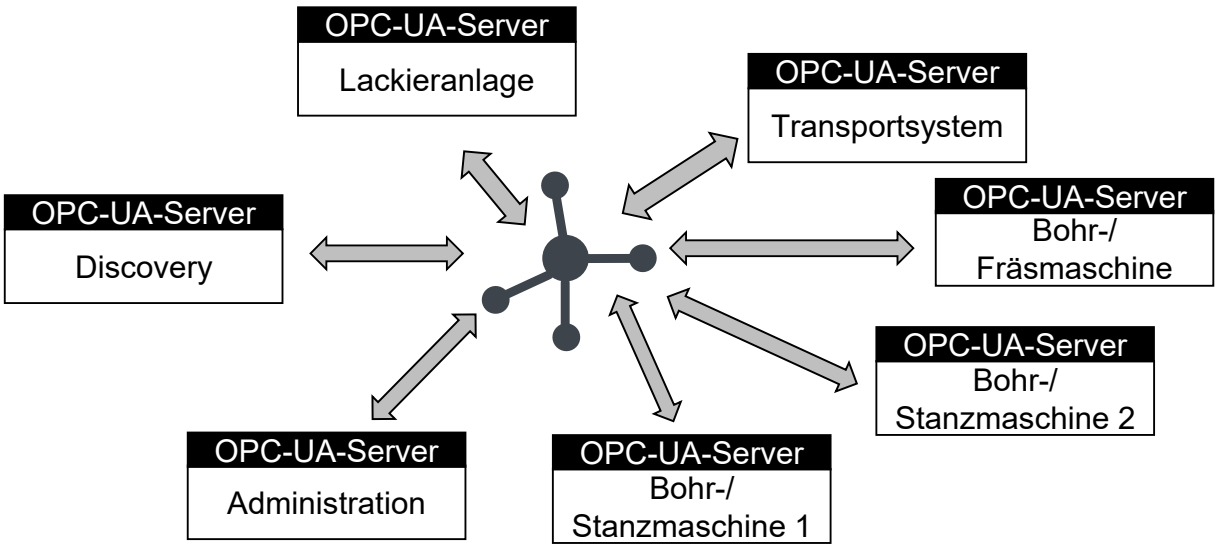


Abbildung 44: Struktur des OPC-UA-Netzwerks

Die OPC-UA-Server der sechs Steuerungen beinhalten insgesamt 111 Services. Die Services kapseln Funktionalitäten des verteilten Steuerungssystems. Sie wurden modellbasiert entwickelt, das heißt sie wurden zuerst als Petri-Netze modelliert und anschließend wurde von den Modellen ausgehend der Programmcode geschrieben. Bei der Modellierung der Petri-Netze wurde die in Kapitel 3.3.2 beschriebene Konvention eingehalten.

Zwischen den Services bestehen 209 Abhängigkeiten. Nach der Fan-in-/Fan-out-Metrik beträgt der durchschnittliche Vernetzungsgrad zwischen den Services 4,06. Die zyklomatische Komplexität der Services erreicht nach der Mc-Cabe-Metrik Werte bis zu 35. Aufgrund der hohen Komplexität des entwickelten Steuerungssystems sind dessen Verhalten und Abhängigkeiten durch manuelle Tätigkeiten kaum analysierbar.

5.2.1 Wiederverwendbarkeit von Services

Die Servicebeschreibung ist so konzipiert, dass ein Service unabhängig von seiner Implementierung einfach in das Steuerungssystem integriert oder in einem neuen Kontext wiederverwendet werden kann. Jeder Service ist über folgende Attribute eindeutig beschrieben:

- Servicename
- Location
- Version
- Hersteller & Typ

Zur Koordination innerhalb des Steuerungssystems sind nur die Attribute „Servicename“ und „Location“ relevant. Der Servicename beschreibt eindeutig die Fähigkeit, die von einem Service angeboten wird. Er ist unabhängig von der Implementierung des Service und erlaubt somit eine einfache Austauschbarkeit von Services verschiedener Hersteller. So bieten beispielsweise alle

Näherungssensoren, gleich welcher Hersteller, den Service „detectObject“ an und können einfach ersetzt werden. Das Topologieattribut „Location“ beschreibt die logische Position eines Service in der Produktionsanlage gemäß der in Kapitel 5.1 beschriebenen Topologie. Somit lässt sich beispielsweise unterscheiden, an welcher Position ein Näherungssensor ist oder welche Bohrmaschine vom Service „drilling“ gesteuert wird. Bei Integration eines Service muss somit nur seine Location angepasst werden. Änderungen an seiner Funktionalität oder Schnittstellen sind nicht notwendig. Die Anpassung der Location wird aktuell durch Einlesen einer Textdatei durchgeführt, kann aber auch über RFID-Tags erfolgen. Die Topologie wurde so entworfen, dass ein Service anhand relativer Beziehungen aus der eigenen Location interpretieren kann, welche Location ein aufzurufender Service besitzen muss. Komponenten, die keine physische Abhängigkeiten besitzen, zum Beispiel höherwertige Steuerungen, besitzen die Location 00:00:00.

Über die Attribute „Hersteller & Typ“ und „Version“ lässt sich die Implementierung des Service eindeutig identifizieren. Da diese zur Koordination des Produktionsablaufs nicht notwendig sind, werden sie vom verteilten Steuerungssystem nicht ausgewertet.

5.2.2 Ad-hoc-Vernetzbarkeit und Rekonfigurierbarkeit

Die OPC-UA-Server registrieren sich bei Integration in das Steuerungsnetzwerk beim Discovery Server und senden zyklisch Keepalive-Nachrichten. Die Information, welche OPC-UA-Server verfügbar sind und welche Services diese anbieten, verwaltet der Discovery-Server in einer Datenbank. Der Datenbankzugriff innerhalb des Discovery-Servers erfolgt über den Java Database Connector (JDBC). Wird ein OPC-UA-Server vom Netzwerk getrennt, löscht der Discovery-Server den entsprechenden Eintrag aus der Datenbank nach einer definierten Zeitdauer.

Zur vollständigen Umsetzung einer Ad-hoc-Integration muss der Neueintritt eines OPC-UA-Servers bei den vorhandenen OPC-UA-Servern bekannt gemacht werden. Diese Bekanntmachung geschieht bei Bedarf. Benötigt ein Service die Funktionalität eines anderen Service, fragt dessen OPC-UA-Client den Discovery-Server an. Dieser gibt eine Liste aktuell verfügbarer Services samt deren Attributen und Netzwerkadressen zurück. Ein neu hinzugefügter Service wird somit bei jeder neuen Anfrage mit aufgeführt und dadurch automatisch eingebunden. Nun obliegt es dem aufrufenden Service, den geeignetsten Service direkt aufzurufen. Beim Entfernen eines Service rekonfiguriert sich das System dergestalt, dass auf einen redundanten Service zurückgegriffen wird.

5.2.3 Mechanismus zur Durchführung von Funktionsänderungen an der Steuerungssoftware

Der Mechanismus zur Änderung einer Steuerungsfunktion ist in Abbildung 45 am Beispiel des Transportsystems dargestellt. Die Funktionsänderungen werden durch die Software in der Betriebsphase verursacht.

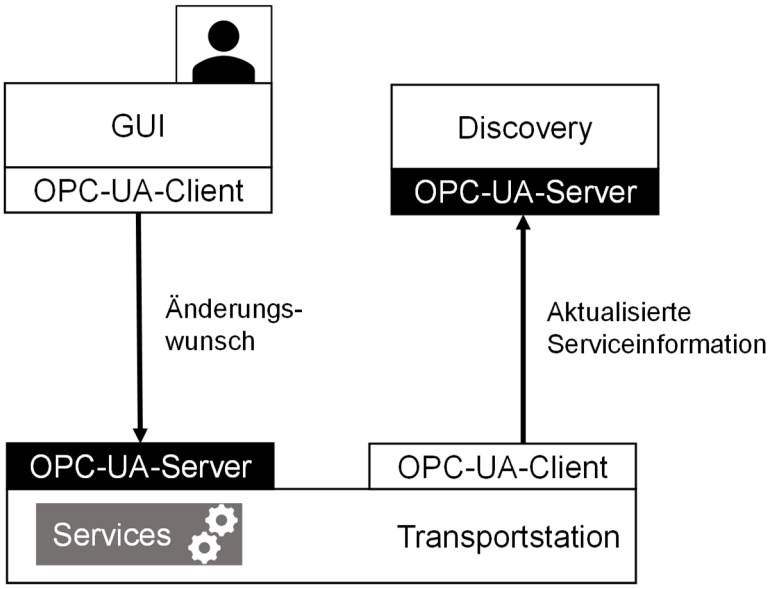


Abbildung 45: Mechanismus zur Änderung der Funktionalität von Services

Für sechs Services des Steuerungssystems sind mehrere Versionen auf den Steuerungen implementiert. Auf diese änderbaren Services wird in Kapitel 6.2.1 eingegangen. Eine graphische Benutzungsschnittstelle (GUI) ermöglicht die Initiierung einer Funktionsänderung. Dieser Änderungswunsch wird mittels eines OPC-UA-Client an den entsprechenden OPC-UA-Server gesendet. Dieser passt seine Service-Schnittstelle so an, dass diese zukünftig die gewünschte Version anbietet und bei Serviceaufruf ausführt. Zur Aktualisierung des Discovery-Servers wird eine aktualisierte Liste der angebotenen Services an ihn gesendet. Funktionsänderungen an einem OPC-UA-Server lassen sich zudem direkt über ein Touch-Display an dessen Host-Einplatinenrechner durchführen.

Zusätzlich zu den beschriebenen Mechanismen lassen sich, durch Trennen und Integration der OPC-UA-Server vom Netzwerk, weitere Änderungsszenarien realisieren. Zur Demonstration der Integration von einem zusätzlichen OPC-UA-Server wurde ein Einplatinenrechner, statt über WLAN, über ein Netzwerkkabel mit dem Steuerungsnetzwerk verbunden.

5.3 Eignung des Automatisierungssystems als Testobjekt

Die verteilte Steuerung des Automatisierungssystems besitzt zahlreiche Eigenschaften, welche zukünftigen Steuerungssystemen zugeschrieben werden:

- verteilte Struktur: Das Steuerungssystem ist auf sechs Rechner verteilt.
- Fähigkeit zur Ad-hoc-Vernetzung: Die Steuerungen lassen sich ohne Konfigurationsaufwand dem Steuerungssystem hinzufügen und aus dem Steuerungsnetzwerk entfernen.
- Software-Änderbarkeit: Über verschiedene Schnittstellen lassen sich Funktionsänderungen an der Software der Steuerungskomponenten unkompliziert durchführen.
- Redundanz: Die implementierten Produktionsmaschinen bieten teilweise gleiche Services an. Weitere Redundanz wird durch die Transporteinheit erreicht. So sind verschiedene Routen zum Erreichen einer Zielposition möglich.
- Wiederverwendbarkeit von Komponenten: Aufgrund einer Topologie lassen sich baugleiche Komponenten mehrfach in das Automatisierungssystem integrieren, ohne dass ein größerer Konfigurationsaufwand notwendig ist.

Größere Allgemeingültigkeit des implementierten Automatisierungssystems wird erreicht, indem der vielversprechende Standard OPC-UA als M2M-Kommunikationsprotokoll verwendet wird. Dazu wird der kommerzielle Java-basierte OPC-UA-Stack des Herstellers Prosys verwendet [133].

Das erstellte Automatisierungssystem beinhaltet für die Automatisierungstechnik typische Charakteristika. So entstehen bei der Steuerung des Produktionsablaufs Parallelitäten und zeitliche Abhängigkeiten. Darüber hinaus müssen Echtzeitanforderungen berücksichtigt werden.

Dabei besitzt das verteilte Steuerungssystem eine Komplexität, welche durch manuelle Analysen kaum zu beherrschen sind.

Aufgrund der Erfüllung der in Kapitel 2.1.2 beschriebenen Eigenschaften zukünftiger Steuerungssysteme und typischen Eigenschaften aktueller Automatisierungssysteme eignet sich das beschriebene Automatisierungssystem als Testobjekt für TestIAS.

6 Evaluierung

In diesem Kapitel wird beschrieben, wie das Konzept anhand dessen Umsetzung empirisch evaluiert wurde. Zur Darlegung der verwendeten Systematik wird zunächst in Kapitel 6.1 in den zur Evaluierung entworfenen Prozess eingeführt. Grundsätzlich wird im Rahmen der empirischen Evaluierung systematisch aufgezeigt, dass durch das vorgelegte Konzept Funktionsänderungen an verteilten Automatisierungssystemen korrekt abgesichert werden können. Dazu werden in Kapitel 6.2 die entworfenen Änderungsszenarien erläutert und es wird aufgezeigt, welche Art der Änderung diese abdecken. Darauf aufbauend wird in Kapitel 6.3 beschrieben, wie genau anhand dieser Änderungsszenarien geprüft wird, ob die Realisierung des Konzepts den definierten Konzept-Anforderungen genügt. Dazu werden die Ergebnisse des Absicherungsprozesses ermittelt, ausgewertet und bewertet.

6.1 Beschreibung des Evaluierungsprozesses

Zur Evaluierung des entworfenen Konzepts wurde ein fünfphasiger Evaluierungsprozess definiert, welcher in Abbildung 46 dargestellt ist.

In Kapitel 4 ist die Realisierung des Konzepts beschrieben, welche prototypisch implementiert wurde. Die Implementierung TestIAS bildet die Grundlage für die empirische Evaluierung des erstellten Konzepts und somit die erste Phase des Evaluierungsprozesses. Die Formulierung der Zielstellung der Evaluierung ist Bestandteil der zweiten Phase des Evaluierungsprozesses. Bei dieser Untersuchung liegt die Zielstellung darin, zu überprüfen, ob die in der Zielsetzung der Arbeit und den Nebenbedingungen definierten Anforderungen erfüllt sind (siehe Kapitel 1.3).

In der dritten Phase „Planung und Aufbau des Evaluierungsprojekts“ wurden die Rahmenbedingungen definiert, welche gegeben sein müssen, um anhand von TestIAS das Konzept zu evaluieren. Darüber hinaus wurde die Evaluierungsumgebung aufgebaut. Zur empirischen Evaluierung wird neben dem Testsystem auch ein Automatisierungssystem benötigt, welches abgesichert werden soll. Dazu wurde das verteilte Automatisierungssystem, das in Kapitel 5 beschrieben ist, als Testobjekt entworfen und implementiert. Bei Entwurf der verteilten Steuerung des Automatisierungssystems wurde dabei berücksichtigt, dass Funktionsänderungen einfach durchzuführen sind. Im Rahmen dieser Phase wurden neun Änderungsszenarien definiert, welche ein repräsentatives Spektrum möglicher Änderungstypen und möglicher daraus resultierender Fehlerarten abdecken. Die zu erwartenden Ergebnisse der Absicherung wurden festgehalten und gelten als Referenz für die in der Evaluierung berechneten Ergebnisse.

Die vierte Phase „Ermittlung und Auswertung der Evaluationsergebnisse“ beschreibt die Durchführung der empirischen Evaluierung sowie die Auswertung der ermittelten Ergebnisse. In dieser

Phase wurden die in der vorherigen Phase definierten Änderungsszenarien am verteilten Automatisierungssystem durchgeführt und deren Auswirkungen mithilfe von TestIAS überprüft. Die Ergebnisse der Verifikation wurden ermittelt und anschließend interpretiert. Die Erfüllung der definierten Zielstellung wurde dabei anhand der interpretierten Ergebnisse sowie anhand theoretischer Betrachtungen diskutiert.

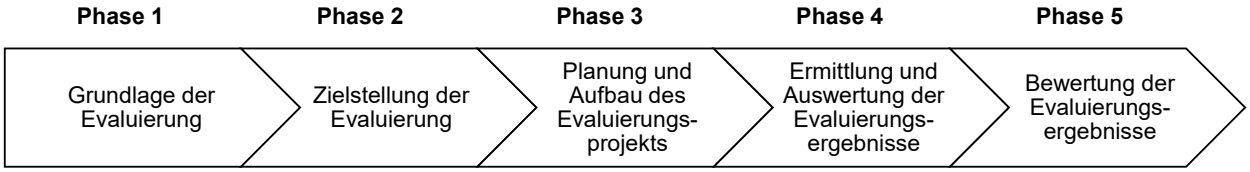


Abbildung 46: Entworfener Evaluierungsprozess angelehnt an [140]

Die Grundlage der Evaluierung (Phase 1), die Zielstellung der Evaluierung (Phase 2) sowie die Planung und der Aufbau der zur Evaluierung notwendigen Objekte (Phase 3) wurden in den vorigen Kapiteln ausgeführt. Somit beschränken sich folgend die Ausführungen der Evaluierung auf offene Aspekte der Phase 3 sowie auf Phase 4 und Phase 5.

6.2 Definition geeigneter Funktionsänderungen

Bei der Planung des Evaluierungsprozesses in Phase 3 wurde analysiert, wie das typische Vorgehen bei Durchführung und Absicherung von Funktionsänderungen ist, welche Arten von Funktionsänderungen an der Steuerungssoftware möglich sind und welche Fehlertypen aus solchen Funktionsänderungen resultieren können. Diese Aspekte werden im Weiteren diskutiert.

Im Kontext dieser Untersuchung werden Funktionsänderungen an Steuerungssystemen während der Betriebsphase betrachtet. So ist davon auszugehen, dass die Anlage initial in Betrieb ist und ihre Funktionalitäten abgesichert sind. Wie in Abbildung 47 illustriert, wird davon ausgegangen, dass der Betrieb zur Durchführung einer Funktionsänderung gestoppt wird. Bevor das Automatisierungssystem nach Durchführung einer Funktionsänderung an einer Steuerungskomponente wieder in Betrieb genommen werden kann, muss das Automatisierungssystem erneut abgesichert werden. Dieser Schritt wird im Rahmen der Evaluierung von TestIAS durchgeführt. TestIAS erkennt von der Funktionsänderung betroffene, funktionale Anforderungen und berechnet über formale Verifikationsverfahren, ob diese noch gültig sind. Bei positivem Ergebnis der Absicherung kann der Betrieb des Automatisierungssystems wiederaufgenommen werden.

Abbildung 47: Ablauf des Prozesses zur Durchführung und Absicherung von Funktionsänderungen

Die im Rahmen der Evaluierung durchgeführten Funktionsänderungen können aus informationstechnischer Sicht folgende Ausprägungen haben:

- Integration eines Service: Dies hat den Zweck, einer Steuerung eine Funktionalität hinzuzufügen. Das kann ein zusätzliches Steuerungsprogramm einer Produktionsmaschine oder ein Steuerungsprogramm für die Produktion eines neuen Produkttyps sein. Bei der Absicherung muss einerseits überprüft werden, ob der neu integrierte Service kompatibel mit den Services ist, auf die er zugreift. Andererseits muss überprüft werden, ob die Services, die auf den neu integrierten Service zugreifen, mit diesem kompatibel sind.

- Änderung eines Service: Bei Änderung der Funktionalität eines Service muss, analog zu „Integration eines Service“, einerseits überprüft werden, ob der geänderte Service mit den Services kompatibel ist auf die er zugreift. Andererseits muss überprüft werden, ob die Services, die auf den geänderten Service zugreifen, noch kompatibel sind.

- Trennung eines Service: Wenn ein Steuerungsprogramm gelöscht wird, muss überprüft werden, ob andere Services von diesem Service abhängig sind und deshalb nicht mehr anforderungsgemäß ausgeführt werden können.

- Integration eines OPC-UA-Servers (mehrere Services): Dies stellt die Integration einer neuen Steuerung, zum Beispiel durch Hinzufügen einer Produktionsmaschine, dar. Analog zu „Integration eines Service“ muss ausgehend von jedem Service des hinzugefügten OPC-UA-Servers einerseits überprüft werden, ob die neu hinzugefügten Services kompatibel mit den Services sind, auf die diese zugreifen. Andererseits muss überprüft werden, ob die Services, die auf die geänderten Services zugreifen, mit diesen kompatibel sind.

- Trennung eines OPC-UA-Servers (mehrere Services): Dies stellt das Entfernen einer Steuerung, zum Beispiel durch Abschalten einer Produktionsmaschine dar. Analog zu „Entfernen eines Service“ muss überprüft werden, ob andere Services von diesen Services abhängig sind und deshalb nicht mehr anforderungsgemäß ausgeführt werden können.

Bei Umsetzung der beschriebenen Änderungstypen können Fehler entstehen, welche unterschiedliche Ausprägungen besitzen:

1. Fehlerwirkung in geändertem Service: Die Funktionsänderung führt dazu, dass funktionale Anforderungen der geänderten Komponente nicht mehr erfüllt sind.

2. Fehlerwirkung in direkt aufrufendem Service: Die Funktionsänderung kann dazu führen, dass funktionale Anforderungen der aufrufenden Komponente nicht mehr erfüllt sind.

3. Fehlerwirkung in entferntem Service: Die Funktionsänderung führt dazu, dass nur funktionale Anforderungen verletzt werden, welche das Verhalten von höherwertigen Services beschreiben.

Die Fehlertypen können in beliebigen Kombinationen vorkommen. Üblicherweise treten, wenn Fehlerwirkung 1 auftritt, auch Fehlerwirkungen 2 und 3 auf. Bei Aufritt von Fehlerwirkung 2 tritt üblicherweise auch Fehlerwirkung 3 auf. Dabei wird eine Fehlerwirkung nur sichtbar, wenn sich die Wirkung des Fehlers auf eine spezifizierte funktionale Anforderung auswirkt.

Um die Anwendbarkeit der Realisierung des Konzepts zu überprüfen, wurden Änderungsszenarien definiert, welche die beschriebenen Änderungstypen und Fehlertypen widerspiegeln. Im Rahmen der Evaluierung wurde das zu erwartende Ergebnis mit den von TestIAS erhobenen Ergebnissen verglichen.

6.2.1 Änderungsszenarien am verteilten Steuerungssystem

Folgend wird auf die neun Änderungsszenarien eingegangen, welche zur Evaluierung entworfen wurden. In Tabelle 7 ist eine Übersicht über die Änderungsszenarien gegeben. Darin wird für jedes Szenario aufgeführt, auf welcher Ebene innerhalb der hierarchischen SOA sich der geänderte Service befindet. So beschreibt Produktsteuerung einen höherwertigen Service, welcher die Reihenfolge zu bearbeitender Produktionsschritte definiert. Prozesssteuerung beschreibt einen mittelwertigen Service, der komplexere Prozesse beschreibt, an denen meist mehrere Ressourcen beteiligt sind. Ressourcensteuerung beschreibt Services, die sehr stark einer Ressource zugeordnet sind und deren Verhalten direkt steuern. Darüber hinaus beschreibt Tabelle 7 um welchen Änderungstyp es sich handelt und was das zu erwartende Ergebnis ist. Für die Bedeutung der verwendeten Begriffe sei auf die Einführung von Kapitel 6.2 verwiesen.

Das erste Änderungsszenario beschreibt die Integration eines Service in das Steuerungssystem durch den ein neuer Produkttyp für die Herstellung eines Inbusschlüssels gesteuert wird. Die Integration des Service stellt eine Änderung der Software des Steuerungssystems dar. Die Steuerung des Service wurde korrekt implementiert und alle Services, auf die der hinzugefügte Service zugreift, sind mit diesem kompatibel. Von dem neu hinzugefügten Service hängen keine weiteren Services ab. Somit müssen ausschließlich die funktionalen Anforderungen dieser Komponente verifiziert werden. Es ist deshalb nicht zu erwarten, dass durch Integration des Service eine funktionale Anforderung verletzt wird.

Das zweite Änderungsszenario beschreibt die Änderung des Service zur Herstellung eines Inbusschlüssels. Um gravierte Inbusschlüssel anzufertigen, wird der Service um den Prozessschritt des Gravierens erweitert. Der zusätzliche Prozessschritt wurde korrekt implementiert. Die funktionalen Anforderungen der Komponente werden dadurch nicht verletzt und es existieren keine weiteren Services, die von dem geänderten Service abhängig sind. Es ist somit nicht zu erwarten, dass durch Änderung des Service eine funktionale Anforderung verletzt wird.

Tabelle 7: Übersicht der Änderungsszenarien

Änderungsszenario Nr.	Name	Ebene des Service	Änderungstyp	Erwartetes Ergebnis
1	Inbusschlüssel (Standard)	Produktsteuerung	Integration	Kein Fehler
2	Inbusschlüssel (graviert)	Produktsteuerung	Änderung	Kein Fehler
3	Transport zu Warenausgang (Version 3)	Prozesssteuerung	Änderung	Kein Fehler
4	Transport zu Warenausgang (Version 2)	Prozesssteuerung	Änderung	Fehlerwirkung in direkten oder entferntem Service
5	Steuerung von Förderband 4	Prozesssteuerung	Änderung	Fehlerwirkung in entferntem Service
6	Näherungssensor an Förderband 3	Ressourcensteuerung	Änderung	Fehlerwirkung in entferntem Service
7	Motor B der Bohr-/Fräsmaschine	Ressourcensteuerung	Änderung	Fehlerwirkung in direkt aufrufendem Service
8	Motor C der Bohr-/Stanzmaschine 1	Ressourcensteuerung	Änderung	Fehlerwirkung in geändertem Service
9	Entfernen der Bohr-/Stanzmaschine 2	Ressourcensteuerung	Trennung	Fehlerwirkung in direkt aufrufendem Service

Das dritte Änderungsszenario beschreibt die Änderung des Service, welcher den Transport eines fertigen Werkstücks von einer beliebigen Position an den Warenausgang steuert. Der Service wird so angepasst, dass die Werkstücke nun zu einem Warenausgang transportiert werden sollen, wel-

cher sich, statt an Förderband 6, nun an Förderband 5 befindet. Die Implementierung wurde korrekt umgesetzt. Da viele Services den geänderten Service verwenden, ist zu erwarten, dass funktionale Anforderungen von zahlreichen Services erneut abgesichert werden müssen. Dabei sollte es zu keiner Verletzung der betroffenen funktionalen Anforderungen kommen.

Das vierte Änderungsszenario beschreibt analog zu dem dritten Änderungsszenario die Änderung des Service, welcher den Transport eines fertigen Werkstücks von einer beliebigen Position an den Warenausgang bei Förderband 5 statt Förderband 6 steuern soll. Die Anpassung der Steuerungslogik wurde aber nicht konsequent durchgeführt. So werden die Werkstücke zwar zum richtigen Förderband transportiert. Beim letzten Prozessschritt von der Mittelposition des Förderbands zum Warenausgang wird fälschlicherweise das ursprüngliche Förderband 6 aktiviert. Somit verbleibt das Werkstück auf Förderband 5. Dies sollte in einer Fehlerwirkung in einem direkt oder entfernt aufrufenden Service resultieren, da der Produktionsprozess nicht erfolgreich abgeschlossen wird.

Das fünfte Änderungsszenario beschreibt die Änderung des Service, der den Transport eines Werkstücks über Förderband 4 steuert. Initial ist die Steuerung des Förderbands so realisiert, dass das Förderband so lange aktiv ist, bis ein Näherungssensor am Ende des Förderbands erkennt, dass ein Werkstück angekommen ist. Nach Ausfall des Näherungssensors wird die Steuerung so geändert, dass das Förderband, anstatt durch den Näherungssensor eventgesteuert, durch einen Timer zeitgesteuert gestoppt wird. Die für den Transport benötigte Zeitspanne wurde zu gering angegeben, weshalb das Werkstück auf Förderband 4 verbleibt. Da der geänderte Service nicht erkennt, ob das Werkstück an der korrekten Position angekommen ist, ist nicht zu erwarten, dass in dem geänderten Service eine funktionale Anforderung verletzt wird. Dagegen ist zu erwarten, dass funktionale Anforderungen von davon abhängigen Services betroffen sind, da das Werkstück nicht bei der gewünschten Produktionsmaschine ankommt.

Das sechste Änderungsszenario beschreibt einen Fehler, der aus Verletzung von Echtzeitanforderungen resultiert. An Förderband 3 unterhalb der Lackieranlage befindet sich ein Näherungssensor. Der Näherungssensor sorgt dafür, dass das Förderband stoppt, wenn sich ein Werkstück, das lackiert werden soll, unterhalb der Lackieranlage befindet. Über ein Software-Update wird in den Service des Näherungssensors eine zusätzliche Funktionalität eingebracht, für welche eine komplexere Signalverarbeitung notwendig ist. Dies führt zu einer Rechenzeit von 4000 ms, bis der Näherungssensor das Erkennen eines Werkstücks zurückmeldet. Die Modellierung dieses Verhaltens als Petri-Netz ist in Abbildung 48 dargestellt. Bei statischen Systemen stellt diese Funktionsänderung kein Problem dar. In dem vorliegenden Fall bewegt das Förderband das Werkstück aber weiter. Anschließend überprüft die Lackieranlage mittels des Näherungssensors, ob das Werkstück da ist. Dies kann nicht detektiert werden, da es sich durch die Latenz des Näherungssensors nicht mehr unter der Lackieranlage befindet. Das führt zu einem Stillstand des Systems. Da für den Näherungssensor keine Echtzeitanforderungen spezifiziert wurden, ist nicht zu erwarten, dass dort ein Fehler erkannt wird. Es ist auch nicht davon auszugehen, dass der Service,

der den Transport koordiniert, einen Fehler erkennt, da dieser Prozess ordnungsgemäß abgeschlossen wird. Erst in höherwertigen Services sollte erkannt werden, dass der Produktionsablauf unterbrochen ist.

Das siebte Änderungsszenario beschreibt ebenso die Verletzung einer Echtzeitanforderung. Motor B bewegt den Fräskopf der Bohr-/Fräsmaschine in horizontaler Richtung. Um den Fräsvorgang für andere Werkstoffe einzustellen und somit bessere Fräsergebnisse zu erhalten, wird das Anlaufverhalten von Motor B angepasst. Durch die daraus resultierende geringe Fräsgeschwindigkeit verlängert sich die Zeitdauer, die zur Durchführung eines Fräsvorgangs notwendig ist. Bei Durchführung der Änderung wurde nicht beachtet, dass die Frässteuerung über eine Sicherheitsfunktion verfügt, die den Fräsvorgang abbricht, wenn er nicht innerhalb einer definierten Zeitspanne abgeschlossen ist. Durch das veränderte Anlaufverhalten ist damit zu rechnen, dass funktionale Anforderungen des Service, der die Frässteuerung beinhaltet, aufgrund der Sicherheitsfunktion nicht mehr eingehalten werden.

Das achte Änderungsszenario beschreibt die Anpassung der Steuerung des Motors C, welcher den Bohrkopf der Bohr-/Stanzmaschine 1 rotieren lässt. Bei Anpassung des Steuerungsprogramms ist ein Fehler unterlaufen, weshalb es zu einer Verklemmung kommt, welche verhindert, dass Motor C aktiviert werden kann. Da dies funktionale Anforderungen des geänderten Service verletzt, ist damit zu rechnen, dass der Fehler direkt bei diesem Service aufritt und sich auf abhängige Services auswirkt.

Das neunte Änderungsszenario beschreibt das Entfernen der Bohr-/Stanzmaschine 2 von der Anlage. Dies resultiert informationstechnisch darin, dass ein OPC-UA-Server mitsamt den von ihm angebotenen Services aus dem Steuerungssystem entfernt wird. Dabei ist zu erwarten, dass die funktionalen Anforderungen aller Services, die auf einen der entfernten Services zugreifen, neu abgesichert werden müssen. Da die Bohr-/Stanzmaschine redundant ist, wird davon ausgegangen, dass die Funktionalitäten der aufrufenden Services noch erfüllt sind.

Zur Veranschaulichung wie Änderungen in den Verhaltensmodellen abgebildet werden, ist in Abbildung 48 beispielhaft die Funktionsänderung des „Näherungssensors Förderband 3“ dargestellt. Wie im dritten Änderungsszenario beschrieben wurde der Service des Näherungssensors so geändert, dass, aufgrund einer erweiterten Signalverarbeitung, die Bestätigung, dass ein Objekt erkannt wurde, um 4000 ms verzögert an den aufrufenden Service zurückgegeben wird.

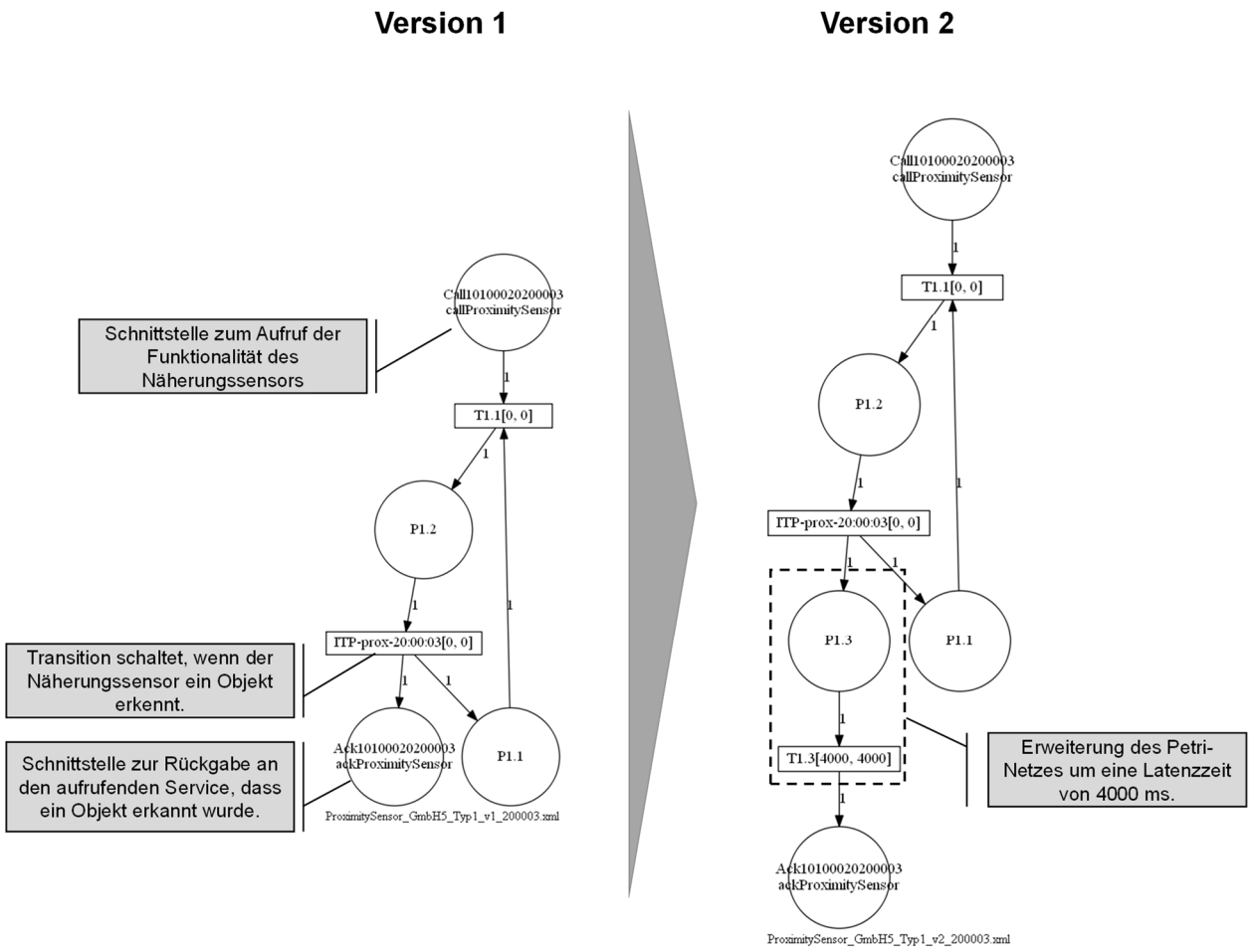


Abbildung 48: Funktionsänderung beim Szenario "Näherungssensor an Förderband 3"

Im Ruhemodus des Näherungssensors aus Abbildung 48 befindet sich ein Token in Stelle *P1.1*. Wird nun der Näherungssensor durch Belegen der Stelle *callProximitySensor* aktiviert, schaltet Transition *T1.1*, die Token der Vorgängerstellen werden abgezogen und in Stelle *P1.2* wird ein Token erzeugt. Wird ein Objekt detektiert, schaltet nun die Transition *ITP-prox-20:00:03*, das Token aus Stelle *P1.2* wird abgezogen und jeweils ein Token werden in Stelle *ackProximitySensor* und Stelle *P1.1* erzeugt. Über die Stelle *ackProximitySensor* wird an den aufrufenden Service zurückgegeben, dass ein Objekt erkannt wurde. Bei Version 2, welche rechts dargestellt ist, wurde das Petri-Netz so geändert, dass nicht beim Erkennen des Objekts direkt ein Token über *ackProximitySensor* an den aufrufenden Service zurückgegeben wird, sondern zuerst in *P1.3* ein Token erzeugt wird. Nach einer Latenzzeit von 4000 ms schaltet Transition *T1.3* und die Bestätigung wird verzögert an den aufrufenden Service zurückgegeben. Dies repräsentiert das Zeitverhalten durch die erweiterte Signalverarbeitung.

Aus Abbildung 48 wird zudem deutlich, dass durch die Modularisierung die Änderung an einem einzelnen Services beherrschbar ist. Die Komplexität steigert sich durch die Vernetzung.

Für eine detaillierte Darstellung der Änderungsszenarien sei auf [138] verwiesen.

6.3 Evaluierung anhand der Änderungsszenarien

Zur Evaluierung wurden die Auswirkungen der in Kapitel 6.2.1 beschriebenen Änderungsszenarien mit TestIAS formal überprüft. Auf die Durchführung der Überprüfung sowie die anschließende Auswertung und Bewertung wird in diesem Kapitel eingegangen.

Zur Durchführung der empirischen Evaluierung wurde das verteilte Steuerungssystem in eine definierte Grundkonfiguration versetzt und anschließend das jeweilige Änderungsszenario durchgeführt. Zur Durchführung der Änderungen an der verteilten Steuerung wurde der in Kapitel 5.2.3 beschriebene Mechanismus verwendet. Nach demselben Prinzip lassen sich einzelne Services hinzufügen und entfernen. Um Software-Änderungen an einem Service nachzubilden, wurden die beschriebenen Änderungen der Services modelliert und implementiert. Wie in Abbildung 48 beispielhaft dargestellt, unterscheiden sich die Versionen in der Funktionalität, also dem Steuerungsablauf. Das Hinzufügen und Entfernen eines OPC-UA-Servers mitsamt seinen Services ist aufgrund der Ad-hoc-Fähigkeit durch physisches Trennen und Verbinden des jeweiligen Einplatinenrechners vom beziehungsweise zum Steuerungsnetzwerk möglich.

Beim Start der Absicherung erkennt TestIAS die Änderungen zur Grundkonfiguration. Die Verifikationsergebnisse sowie die Zeitmessungen der Absicherungsprozesse wurden erfasst und interpretiert. Zur Evaluierung wurde der in Kapitel 4.6.3 beschriebene PC verwendet.

6.3.1 Ermittlung und Auswertung der Evaluierungsergebnisse

Die Ergebnisse der Evaluierung sind in Tabelle 8 aufgeführt. Bei acht von neun Änderungsszenarien entsprach das erwartete Ergebnis dem Ergebnis von TestIAS. Weitergehend wurde durch Ausführung der Steuerung anhand der Simulation überprüft, ob das Automatisierungssystem dem erwarteten Verhalten entspricht.

Tabelle 8: Ergebnisse der Absicherung nach Änderungen

Änderungsszenario	**Rechendauer**	**Anzahl zu verifizierender Komponenten**	**Verifikation erwartungsgemäß?**
Inbusschlüssel (Standard)	1 h 11 min	1	✓
Inbusschlüssel (graviert)	2 h 49 min	1	✓
Transport zu Warenausgang (Version 3)	7 h 11 min	11	✓
Transport zu Warenausgang (Version 2)	7 h 30 min	11	×
Steuerung von Förderband 4	6 h 15 min	22	✓
Näherungssensor an Förderband 3	4 h 50 min	15	✓
Motor B der Bohr-/Fräsmaschine	1 h 56 min	6	✓
Motor C der Bohr-/Stanzmaschine 1	8 h 22 min	6	✓
Entfernen der Bohr-/Stanzmaschine 2	3 h 41 min	11	✓

Die Verifikation der Anforderungen aller 111 Services dauert auf dem verwendeten PC 9 h 45 min. Durch die Auswirkungsanalyse konnten die betroffenen Services auf durchschnittlich 10,3 eingegrenzt und somit eine Reduktion um 91 % erreicht werden. Dadurch verringerte sich die notwendige Dauer zur Absicherung um 50 % auf durchschnittlich 4 h 51 min. Dies ist in Abbildung 49 und Abbildung 50 graphisch verdeutlicht.

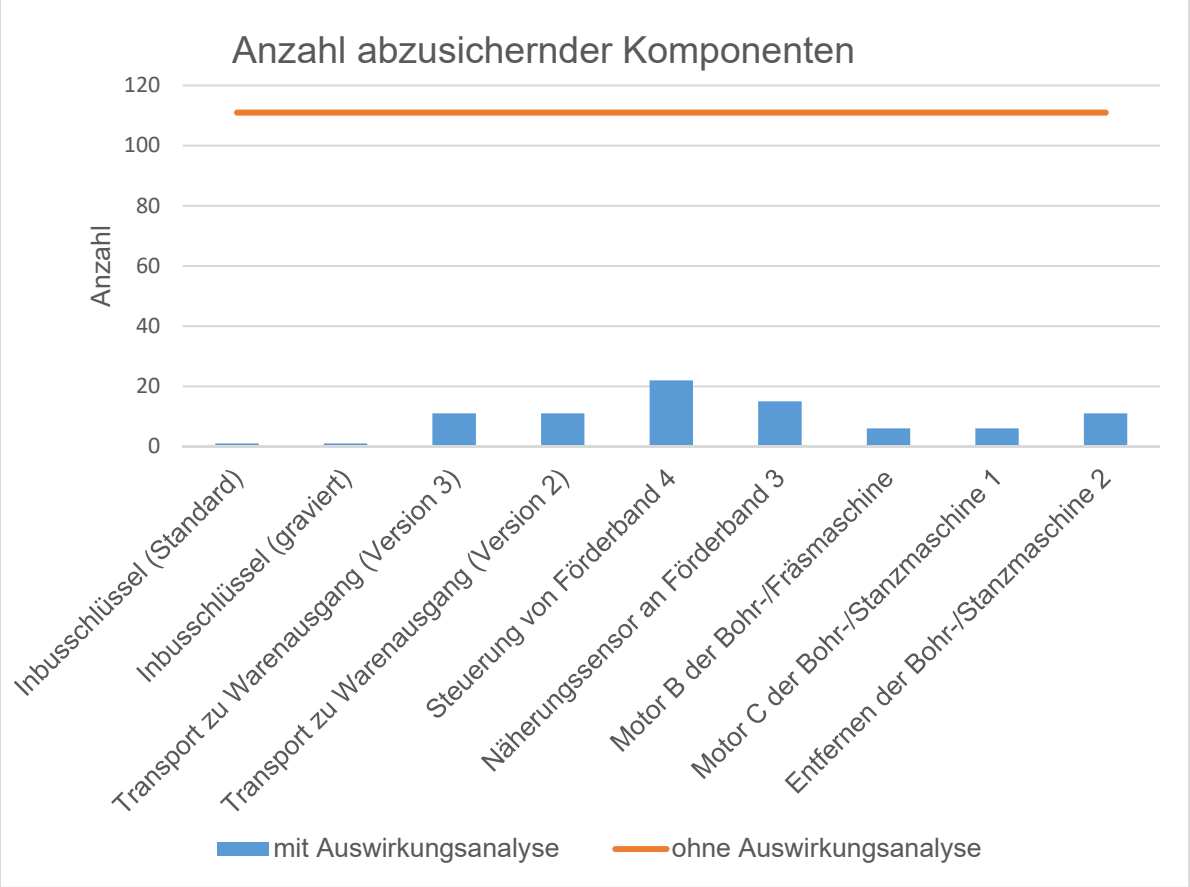


Abbildung 49: Anzahl der Komponenten, welche bei den Änderungsszenarien erneut abgesichert werden müssen

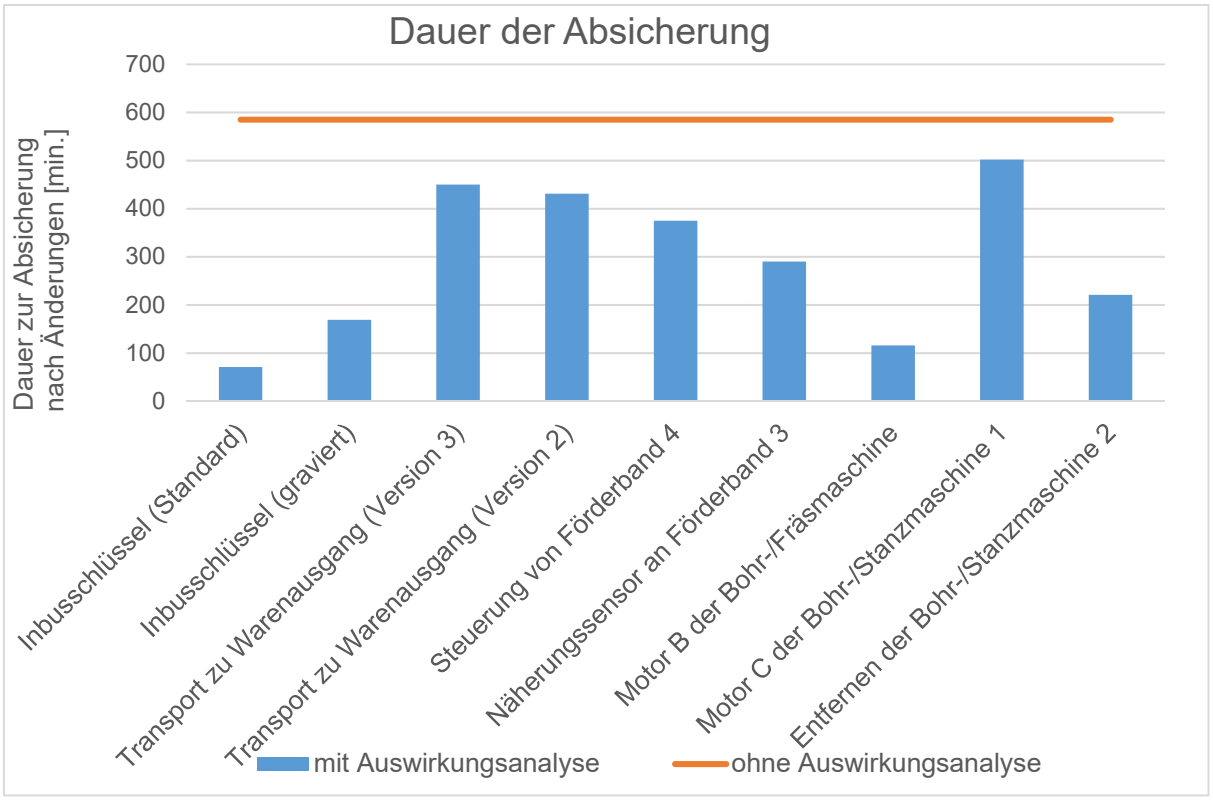


Abbildung 50: Verifikationsdauer der Änderungsszenarien

Um zu überprüfen, ob die funktionalen Anforderungen aller Services betrachtet wurden, die von dem geänderten Service abhängig sind, wurden die Ergebnisse der Auswirkungsanalyse mit den Erwartungen verglichen, die bei Erstellung der Änderungsszenarien spezifiziert wurden. Zum Abgleich der betroffenen Services wurde das von TestIAS generierte Blockdefinitionsdiagramm verwendet. Dies ist exemplarisch für das sechste Änderungsszenario „Näherungssensor an Förderband 3“ in Abbildung 51 dargestellt. Die Abschnitte des Blockdefinitionsdiagramms, welche von der Funktionsänderung betroffene Services enthalten, sind vergrößert dargestellt. Die betroffenen Services werden im Blockdefinitionsdiagramm hellblau hervorgehoben.

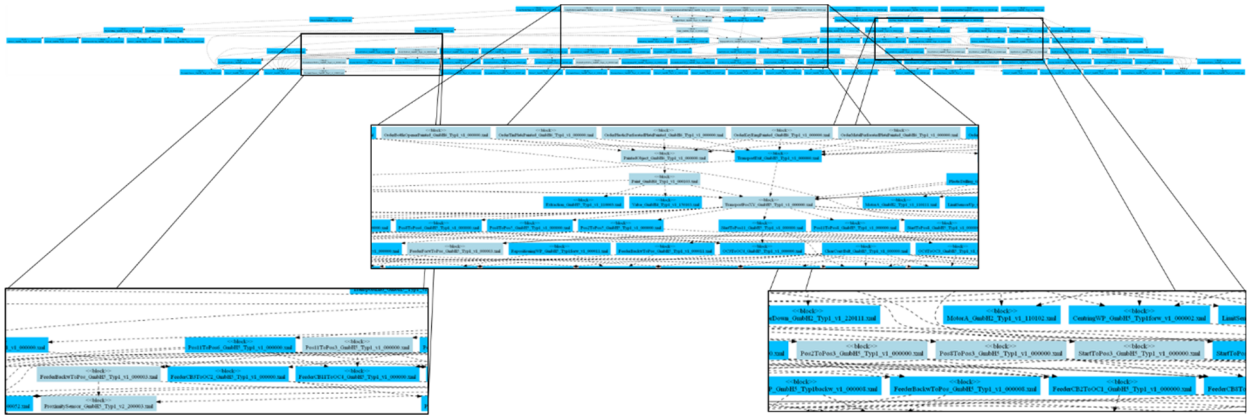


Abbildung 51: Blockdefinitionsdiagramm, welches bei Absicherung der Funktionsänderung beim Szenario „Näherungssensor an Förderband 3“ generiert wurde

Die Gültigkeit der Auswirkungsanalyse konnte durch stichprobenartige Überprüfungen bestätigt werden. So konnte empirisch gezeigt werden, dass die funktionalen Anforderungen, welche durch die Auswirkungsanalyse nicht als kritisch eingestuft wurden, auch nach den Funktionsänderungen noch erfüllt sind.

6.3.2 Bewertung der Evaluierungsergebnisse

Anknüpfend an die Ergebnisse wird deren Aussagekraft im Rahmen der sechsten Phase des Evaluierungsprozesses beleuchtet. Dazu wird über die Verifikationsdauer der Verhaltensmodelle die Skalierbarkeit betrachtet. Anschließend wird anhand der Ergebnisse dargestellt, welche Fehlertypen von TestIAS erkannt werden können. Abschließend wird die Erfüllung der in Kapitel 1.3 definierten Zielsetzung diskutiert.

6.3.2.1 Verifikationsdauer eines Petri-Netzes

Die benötigten Zeitdauern zur Verifikation aller implementierten Services des verteilten Steuerungssystems sind in Abbildung 52 dargestellt. Die Zeitmessung wurde auf dem in Kapitel 4.6.3 beschriebenen PC mit dem Model Checker ITS-Tools durchgeführt.

Die zur Verifikation eines Petri-Netzes benötigte Zeitdauer in Abhängigkeit von der Anzahl Stellen ähnelt einem exponentiellen Verlauf. Diese Messungen machen deutlich, wie elementar wichtig die Eingrenzung betroffener funktionaler Anforderungen durch Auswirkungsanalysen und die Komposition ausschließlich zur Verifikation notwendiger Petri-Netze beim Einsatz von Verifikationswerkzeugen sind.

Aufgrund der Kapselung von Steuerungssystemen ist aber nicht zu erwarten, dass die Anzahl der Stellen und die Komplexität des Verhaltensmodells eines Steuerungssystems ins nicht mehr Handhabbare wachsen. Das größte aus der Komposition entstandene Verhaltensmodell verfügt, nach der Entfaltung, über 546 Stellen und benötigte zur Verifikation 5 h 17 min.

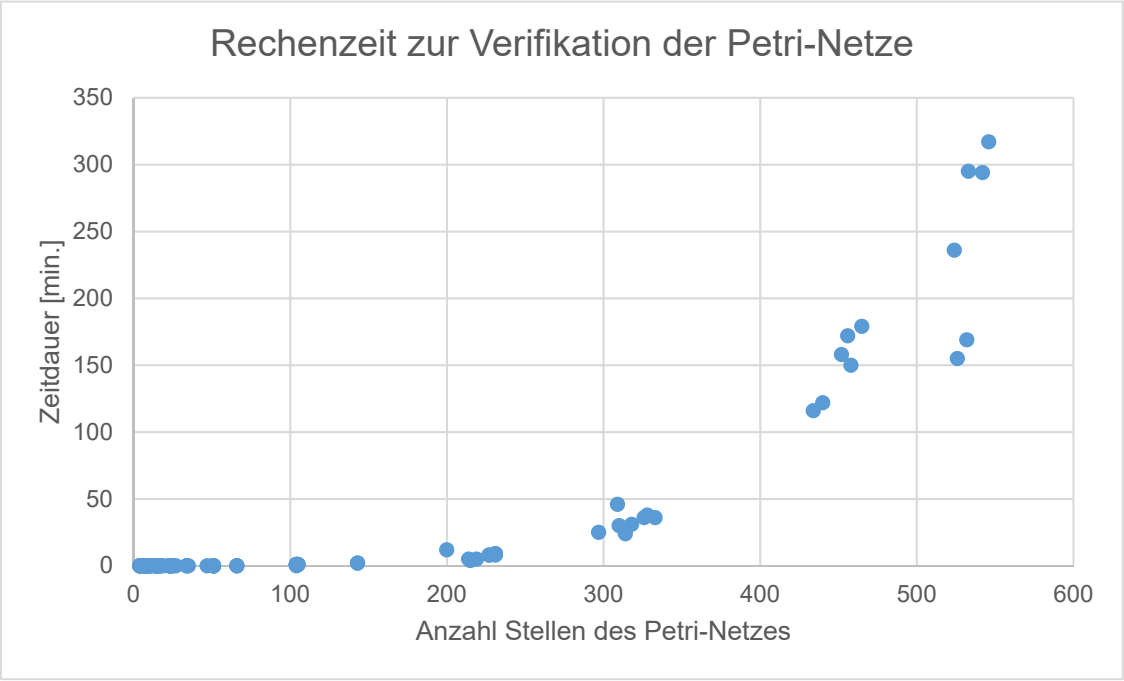


Abbildung 52: Darstellung der Verifikationsdauer aller 111 Petri-Netze in Abhängigkeit von der Anzahl der Stellen des Petri-Netzes

6.3.2.2 Erkennbare Fehlertypen

Die bei der Verifikation erkennbaren Fehlertypen hängen stark von folgenden Faktoren ab:

Verhaltensmodellierung:

Verhaltensmodelle bilden die Basis für das beschriebene Konzept. Welches Verhalten analysiert werden kann, hängt stark von der Sicht und der Mächtigkeit der Modellierungsart ab. Die gewählte Modellierungsart „Petri-Netze“ erlaubt die Beschreibung von diskreten Abläufen. Diese Abläufe können parallele Prozesse und zeitliches Verhalten aufweisen. Darüber hinaus lässt sich das Verhalten bei Redundanz und Mehrfachzugriff überprüfen.

Neben der Modellierungsart ist die Modellierungstiefe entscheidend. Je detaillierter ein System modelliert ist, desto feingranularer lassen sich Abweichungen des Ist-Verhaltens vom Soll-Verhalten erkennen. Dieser Gewinn an Aussagekraft muss über einen höheren Modellierungsaufwand sowie höhere Verifikationszeiten erkauft werden.

So muss das Ist-Verhalten eines Systems ausreichend über die Verhaltensmodelle abgebildet werden, um ein aussagekräftiges Ergebnis zu erhalten. Um das Soll-Verhalten eines Systems maschinell interpretierbar beschreiben zu können, ist eine formale Anforderungsspezifikation essentiell.

Formale Anforderungsspezifikation:

Bei der formalen Verifikation können nur Fehlertypen erkannt werden, deren Fehlerwirkung sich auf eine Anforderung auswirkt, welche durch die formale Anforderungsspezifikation beschrieben werden kann. Somit beeinflusst die Wahl der Spezifikationssprache direkt die erkennbaren Fehlertypen. Bei der Realisierung des Konzepts wurde die Spezifikationssprache CTL gewählt. Damit lassen sich hauptsächlich folgende Fehlertypen identifizieren:

- Verklemmungen
- Erreichbarkeit von Zuständen (global oder innerhalb definierter Schrittzahl oder Zeit)
- Nichterreichbarkeit von Zuständen (global oder innerhalb definierter Schrittzahl oder Zeit)
- ungültige Zustände
- ungültige Schaltfolgen
- Irreversibilität

Darüber hinaus gibt es konzeptbedingte Rahmenbedingungen, die berücksichtigt werden müssen.

Wie in Kapitel 3.5.4.2 erläutert, besteht die Annahme, dass bei der Modellierung und Spezifikation einer Komponente nicht bekannt ist, mit welchen Komponenten diese im Betrieb interagiert. Somit ist nur der Aufbau der jeweiligen Komponente bekannt. Das resultiert darin, dass sich die Komponenten-Anforderung ausschließlich auf das Verhalten dieser Komponente beziehen kann. So kann nicht als Anforderung definiert werden, in welchem Zustand sich eine aufgerufene Komponente befinden muss.

Das ist auch der Grund dafür, dass bei dem vierten Änderungsszenario „Transport Warenausgang (Version 2**)**“ der Fehler nicht erkannt werden konnte. In diesem Szenario wird Förderband 5 in die falsche Richtung angesteuert. Da dies von keinem Sensor erfasst wird, hat dies keine Rückwirkung auf den Zustand der Steuerung und kann somit auch nicht als Fehler identifiziert werden. Gäbe es durch einen Sensor eine Rückmeldung über die Fahrtrichtung des Förderbands, wäre dieser Fehler identifizierbar gewesen.

6.3.2.3 Erfüllung der Zielsetzung zur Beantwortung der Forschungsfrage

Die Zielsetzung dieser Arbeit besteht darin, eine Antwort auf folgende Forschungsfrage zu geben:

„Wie können nach funktionalen Änderungen an der Steuerungssoftware verteilter Automatisierungssysteme automatisiert Fehler erkannt werden?“

Die Antwort auf diese Frage liefert das in Kapitel 3 vorgelegte Konzept. Dieses Konzept wurde realisiert und empirisch evaluiert. Dabei wurde seine Tauglichkeit für die in Kapitel 6.3.2.2 beschriebenen Fehlertypen bewiesen. Darüber hinaus wurden, zur Sicherstellung der Anwendbarkeit des Konzepts für Anlagenbetreiber, mehrere Nebenbedingungen berücksichtigt. Dies wird im Folgenden dargelegt.

Berücksichtigung der Rahmenbedingungen zukünftiger Automatisierungssysteme: In Kapitel 5.2 wurde verdeutlicht, dass das zur Evaluierung verwendete Steuerungssystem elementare Grundmechanismen und -strukturen zukünftiger Automatisierungssysteme beinhaltet. Dies wurde exemplarisch an einem Steuerungssystem aufgezeigt, kann aber aufgrund der verwendeten Standards in weiten Bereichen generalisiert werden.

Einfache Integration in ein Steuerungssystem: TestIAS nutzt die Ad-hoc-Fähigkeit zukünftiger Steuerungssysteme, damit es ohne vorangehende Konfiguration in das Steuerungssystem integriert und in Betrieb genommen werden kann. Im Rahmen dieser Umsetzung des vorgelegten Konzepts wurde exemplarisch aufgezeigt, welche Informationen für das beschriebene Konzept relevant sind. Des Weiteren wurde dargestellt, dass diese Information ohne tiefe Integration in ein Steuerungsnetzwerk daraus abrufbar sind.

Anwendbarkeit in der Betriebsphase: Die zur Evaluierung erstellten Änderungsszenarien spiegeln realistische Anwendungsfälle in der Betriebsphase wider. Des Weiteren zeigt die Umsetzung des Konzepts auf, dass der Absicherungsprozess das Automatisierungssystem nur wenig beeinflusst. Somit ist bei Durchführung der Absicherung ein direkter Zugriff auf Aktorik, Sensorik sowie die einzelnen Steuerungen nicht zwingend notwendig. Ein Zugriff auf einen Discovery-Server kann ausreichen, um genügend Informationen über die aktuelle Konfiguration des Steuerungssystems zu erlangen.

Hohe Effizienz: Die Umsetzung des Konzepts verdeutlicht, dass der im Konzept beschriebene Absicherungsprozess für den Anlagenbetreiber automatisiert realisiert werden kann. Dies erspart manuellen Aufwand bei der Identifikation erneut abzusichernder Teilsysteme und der manuellen Ausführung von Testfällen.

Für die Verifikation der Anforderungen aller Services des beschriebenen verteilten Automatisierungssystems benötigt TestIAS 9 h 45 min. Durch die Auswirkungsanalyse ließ sich diese Zeit-

dauer für die definierten Änderungsszenarien durchschnittlich um 50 % reduzieren. Diese Verifikation geschieht vollautomatisiert, weshalb in dieser Zeit kein Personalaufwand zur Interaktion und Überwachung notwendig ist.

Einfache Nachvollziehbarkeit: Zur Anwendbarkeit des Konzepts ist eine einfache Nachvollziehbarkeit der Verifikationsergebnisse unerlässlich. Eine Möglichkeit, die Ergebnisse gewinnbringend mit einer graphischen Benutzungsschnittstelle aufzuarbeiten, ist in Kapitel 4.6.1 aufgezeigt. So ist es möglich, Abhängigkeiten innerhalb des Systems darzustellen, Fehlerpfade zu lokalisieren und mögliche Fehlerquellen einzugrenzen.

Somit konnte durch das Konzept eine Antwort auf die Forschungsfrage gefunden und die definierte Zielsetzung erfüllt werden. Diese genügt dabei den genannten Nebenbedingungen. Es konnten dadurch für den Absicherungsprozess von Funktionsänderungen an verteilten Steuerungssystemen Verbesserungen in Zeit, Kosten und Komplexität erreicht werden.

7 Schlussbetrachtungen

7.1 Zusammenfassung der Ergebnisse und Bewertung

Die steigende Bedeutung von Software und die Zunahme von Funktionsänderungen an der Steuerungssoftware von Automatisierungssystemen erfordern einen strukturierten und umfassenden Software-Testprozess. Insbesondere während der Betriebsphase eines Automatisierungssystems gewinnt der Softwaretest zur Absicherung nach Funktionsänderungen an Relevanz. Dies betrifft hauptsächlich Anlagenbetreiber, die traditionell wenig mit dem Thema Softwaretest vertraut sind.

Formale Verifikationsmethoden, wie das Model Checking, können bei der Absicherung von Automatisierungssystemen unterstützen. Vorteilhaft am Model Checking ist die vollautomatisierte Ausführbarkeit und die Beweisbarkeit der Erfüllung von funktionalen Anforderungen. In der Praxis werden formale Verifikationsmethoden aber häufig nicht eingesetzt, da die Erstellung formaler Verhaltensmodelle sehr komplex und fehleranfällig ist und somit hohe Expertise erfordert. Dies stellt den Ausgangspunkt der vorliegenden Arbeit dar.

Um dieser Herausforderung zu begegnen und die Erzeugung formaler Verhaltensmodelle zu erleichtern, wurde in dieser Arbeit ein Konzept zur Absicherung verteilter Automatisierungssysteme entwickelt. Das Konzept gibt vor, wie ausgehend von einer Änderung an der Steuerungssoftware eines verteilten Automatisierungssystems die funktionalen Anforderungen identifiziert werden, deren Gültigkeit nicht mehr gewährleistet werden kann und die deshalb mittels des Model Checking überprüft werden müssen. Die dazu notwendigen formalen Verhaltensmodelle werden automatisiert generiert. Die Eckpfeiler dieses Konzepts werden im Folgenden vorgestellt.

Im Rahmen des entworfenen modularen Modellierungsansatzes wird die Annahme getroffen, dass von jeder Steuerungskomponente ein formales Verhaltensmodell sowie formalisierte funktionale Anforderungen vorliegen. Das Verhaltensmodell beschreibt das Verhalten der Steuerungskomponente und die funktionalen Anforderungen beschreiben die Eigenschaften, welchen die von der Steuerungskomponente angebotenen Funktionalitäten genügen müssen. Entsprechend dem modularen Ansatz können diese Elemente für jede Steuerungskomponente isoliert betrachtet werden.

Zur Modellierung des Verhaltens der Steuerungskomponenten wurde eine für das Konzept geeignete Modellierungsart definiert, welche auf Petri-Netzen beruht. Auf Basis dieser Modellierungsart wurden Prinzipien zur Erzeugung des Verhaltensmodells eines Teilsystems definiert. Diese beruhen auf Kompositionsregeln zur Verknüpfung der Verhaltensmodelle der Steuerungskomponenten und einem konzipierten Verfahren zur Berücksichtigung von Mehrfachzugriff und Redundanz innerhalb der komponierten Verhaltensmodelle. Mehrfachzugriff liegt dann vor, wenn mehrere Komponenten auf die Fähigkeit einer Komponente zugreifen können. Zur Gewährleistung

einer validen Darstellung des Mehrfachzugriffs wurde ein Verfahren entwickelt, mit welchem die Verhaltensmodelle automatisiert zu farbigen Petri-Netzen erweitert werden können.

Diese Modellierungsart und Modelloperationen bilden die Grundlage für den Absicherungsprozess. Zur Identifikation der Auswirkungen einer Software-Änderung wurde ein Verfahren entworfen, das beschreibt, wie anhand der Verhaltensmodelle die Abhängigkeitsbeziehungen zwischen den Steuerungskomponenten analysiert werden können. Aus diesen Abhängigkeitsbeziehungen wird ein abstrakter Abhängigkeitsgraph in Form eines Blockdefinitionsdiagramms generiert. Mithilfe des Blockdefinitionsdiagramms lassen sich die funktionalen Anforderungen der Steuerungskomponenten ermitteln, deren Gültigkeit nach der Funktionsänderung nicht mehr gewährleistet werden kann.

Zur Verifikation dieser funktionalen Anforderungen mithilfe des Model Checking sind Verhaltensmodelle der Steuerungskomponenten notwendig, die zur Erfüllung der funktionalen Anforderungen beitragen. Anhand des Blockdefinitionsdiagramms werden die dazu benötigten Verhaltensmodelle erkannt. Die benötigten Verhaltensmodelle sind nach den konzipierten Modelloperationen automatisiert komponierbar. Die komponierten Verhaltensmodelle lassen sich anschließend gegen die korrespondierenden funktionalen Anforderungen verifizieren. Zur Verifikation können, aufgrund der Einhaltung klassischer Petri-Netz-Konventionen, zahlreiche konventionelle Model Checker verwendet werden.

Anschließend wird eine mögliche Realisierung des Konzepts vorgestellt, welche im Rahmen dieser Arbeit implementiert wurde. Um die Funktionsweise der Realisierung demonstrieren zu können, wurde zusätzlich ein verteiltes Automatisierungssystem als Testobjekt entworfen und umgesetzt. An der Steuerungssoftware dieses Automatisierungssystems können Funktionsänderungen durchgeführt werden.

Mithilfe der Realisierung und dem verteilten Automatisierungssystem wird das Konzept sodann empirisch evaluiert. Dazu werden neun Funktionsänderungen an dem verteilten Automatisierungssystem durchgeführt. Diese Funktionsänderungen decken Ad-hoc-Integration, Ad-hoc-Entfernung oder Änderungen an Steuerungskomponenten ab. Ob das geänderte Automatisierungssystem nach den Funktionsänderungen seinen funktionalen Anforderungen genügt, wird anschließend durch die Realisierung des Konzepts analysiert und verifiziert.

Im Rahmen der Evaluierung wird gezeigt, dass die Identifikation betroffener funktionaler Anforderungen und die Komposition der benötigten Verhaltensmodelle funktioniert. Dabei können aufgrund der konzipierten Modelloperationen Mehrfachzugriff und Redundanz korrekt im Verhaltensmodell abgebildet werden.

Es kann gezeigt werden, dass durch den Absicherungsprozess eine, für Anlagenbetreiber vollautomatisierte, Aussage darüber generierbar ist, ob nach einer Änderung der Steuerungssoftware die funktionalen Anforderungen an das Automatisierungssystem erfüllt sind.

Anhand der Auswirkungsanalyse können die funktionalen Anforderungen, deren Gültigkeit nach einer Änderung der Steuerungssoftware nicht mehr sichergestellt ist, eingegrenzt werden. Die Evaluierung zeigt, dass nicht betroffene Anforderungen korrekt ausgeschlossen werden können und dies in einer Reduktion der für das Model Checking benötigten Rechenzeit resultiert.

7.2 Ausblick

Im Rahmen der vorgenommenen Untersuchung, wurden Herausforderungen identifiziert und adäquate Lösungsansätze entwickelt. Auf Grundlage der gewonnenen Erfahrungen wurden im Laufe des Projekts einige Themenbereiche identifiziert, die Ansätze für die Fortsetzung dieser Arbeit bieten. Das in dieser Arbeit vorgestellte Konzept stellt dafür die Basis dar.

Die Wahl einer Granularität und die Mächtigkeit einer Modellierungsart sind stark abhängig vom Anwendungszweck. Je detaillierter die Modellierung, desto aussagekräftiger ist das Verifikationsergebnis. Dies resultiert aber in einer deutlich höheren Komplexität bei der Modellierung, was zu neuen Fehlerquellen führen kann. Bisher wurde zur Verhaltensmodellierung eine möglichst einfach nachvollziehbare Modellierungsart gewählt. Durch Erweiterung der beschriebenen Petri-Netze um zusätzliche Attribute ließe sich, beispielsweise mittels farbiger Petri-Netze, der Datenaustausch zwischen den Komponenten detaillierter beschreiben. Darüber hinaus ließe sich durch Verwendung einer anderen Modellierungsart, wie Net Condition/Event Systems (NCES), neben der asynchronen Kommunikation auch die synchrone Kommunikation berücksichtigen. Maßnahmen dieser Art können die Güte der Modellierung sowie die Vielfalt der darstellbaren Mechanismen weiter steigern.

Im Fokus dieser Arbeit stehen Funktionsänderungen an der Steuerungssoftware. Aus diesem Grund wurde der technische Prozess nur zweitrangig betrachtet. Eine zusätzliche Berücksichtigung von Änderungen des technischen Prozesses kann durch eine Erweiterung des bestehenden Konzepts erreicht werden. Darüber hinaus kann eine Konzeption zur flexibleren Anbindung der Verhaltensmodelle des Steuerungssystems an die Verhaltensmodelle des technischen Prozesses angestrebt werden. Eine Möglichkeit der flexiblen Anbindung wird im Realisierungsteil dieser Arbeit mithilfe einer Topologie bereits exemplarisch aufgezeigt.

Aufgrund des Beweischarakters wird im Rahmen des Konzeptes das Model Checking verwendet. Statt der Nutzung des Model Checking, welches ein statisches Testverfahren ist, ließen sich auch dynamische Tests zur Absicherung eines Steuerungssystems durchführen. Dabei schafft die Modellkomposition eine Model-in-the-Loop-Umgebung. Zur Durchführung eines dynamischen Testfalls müssten, anstatt formaler Anforderungen, für die Komponenten ausführbare Testfälle spezifiziert werden.

Somit können die in diesem Konzept beschriebenen Modelloperationen als Grundgerüst für zahlreiche weitere Forschungsbestrebungen im Themenbereich Test und Simulation dienen.

Literaturverzeichnis

[1] DIN VDE 0105-100, ‘Betrieb von elektrischen Anlagen’, Oct. 2015.

[2] F. Bitsch, *Verfahren zur Spezifikation funktionaler Sicherheitsanforderungen für Automatisierungssysteme in Temporallogik*. Aachen: Shaker, 2007.

[3] VDI/VDE GMA, ‘Statusreport: Industrie 4.0 Begriffe/Terms’, Oct. 2017.

[4] International Software Testing Qualifications Board (ISTQB/GTB), ‘Standardglossar der Testbegriffe’, Mar. 2014.

[5] H. A. ElMaraghy, ‘Flexible and reconfigurable manufacturing systems paradigms’, *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 4, pp. 261–276, Oct. 2005.

[6] W. T. Councill and G. T. Heineman, *Component-based software engineering: putting the pieces together*. Boston, Mass.: Addison-Wesley, 2001.

[7] P. Linder, *Constraintbasierte Testdatenermittlung für Automatisierungssoftware auf Grundlage von Signalflussplänen*. Aachen: Shaker, 2008.

[8] VDI/VDE GMA, ‘Statusreport: Industrie 4.0 – Technical Assets Grundlegende Begriffe, Konzepte, Lebenszyklen und Verwaltung’, Nov. 2015.

[9] VDI/VDE-Richtlinie 2180 Blatt 1, ‘Funktionale Sicherheit in der Prozessindustrie Einführung, Begriffe, Konzeption’, Feb. 2018.

[10] International Software Testing Qualifications Board (ISTQB), ‘Certified Tester Foundation Level Syllabus, Version 2018’, 2018.

[11] DIN SPEC 91345, ‘Referenzarchitekturmodell Industrie 4.0 (RAMI4.0)’, Apr. 2016.

[12] G. Bengel, *Grundkurs Verteilte Systeme*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014.

[13] K. Thramboulidis, ‘The 3+1 SysML View-Model in Model Integrated Mechatronics’, *Journal of Software Engineering and Applications*, vol. 03, no. 02, pp. 109–118, 2010.

[14] VDI/VDE 2184, ‘Zuverlässiger Betrieb und Wartung von Feldbussystemen’, Jun-2008.

[15] B. Vogel-Heuser, J. Folmer, and C. Legat, ‘Anforderungen an die Softwareevolution in der Automatisierung des Maschinen- und Anlagenbaus’,*at – Automatisierungstechnik*, vol. 62, no. 3, Jan. 2014.

[16] Forschungsunion, ‘Bericht der Promotorengruppe Kommunikation - im Fokus: Das Zukunftsprojekt Industrie 4.0 Handlungsempfehlungen zur Umsetzung’, Mar. 2012.

[17] G. Reinhart, Ed., *Handbuch Industrie 4.0: Geschäftsmodelle, Prozesse, Technik*. München: Carl Hanser Verlag GmbH & Co. KG, 2017.

[18] M. Weyrich *et al.*, ‘Evaluation Model for Assessment of Cyber-Physical Production Systems’, *Industrial Internet of Things: Cybermanufacturing Systems*, pp. 169–199, 2017.

[19] NAMUR NE 148, ‘Automation Requirements relating to Modularisation of Process Plants’, Oct. 2013.

[20] B. Vogel-Heuser *et al.*, ‘Agentenbasierte cyberphysische Produktionssysteme’, atp e*dition – Automatisierungstechnische Praxis, no 57*, 2015.

[21] B. Vogel-Heuser, C. Diedrich, and M. Broy, ‘Anforderungen an CPS aus Sicht der Automatisierungstechnik’, *at – Automatisierungstechnik*, vol. 61, no. 10, Jan. 2013.

[22] H.-P. Wiendahl *et al.*, ‘Changeable Manufacturing - Classification, Design and Operation’, *CIRP Annals*, vol. 56, no. 2, pp. 783–809, 2007.

[23] H. A. ElMaraghy and T. AlGeddawy, ‘Co-evolution of products and manufacturing capabilities and application in auto-parts assembly’, *Flexible Services and Manufacturing Journal*, vol. 24, no. 2, pp. 142–170, Jun. 2012.

[24] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, ‘Evolution of software in automated production systems: Challenges and research directions’, *Journal of Systems and Software*, vol. 110, pp. 54–84, Dec. 2015.

[25] J. Ladiges, A. Fay, and W. Lamersdorf, ‘Automated Determining of Manufacturing Properties and Their Evolutionary Changes from Event Traces’, *Intelligent Industrial Systems*, vol. 2, no. 2, pp. 163–178, Springer, Jun. 2016.

[26] ‘Tesla: Elon Musk meldet Fortschritte bei Tesla-Model-3-Produktion’, *manager magazin*. [Online]. Available: http://www.manager-magazin.de/unternehmen/autoindustrie/tesla-elektroauto-bauer-uebertrifft-erwartungen-elon-musk-genervt-a-1205922.html. [Accessed: 24-Aug-2018].

[27] J. Fisch and C. Diedrich, ‘Methodische Untersuchung des Komplexitätsanstiegs von Produktionssystemen’, *at – Automatisierungstechnik*, vol. 66, 2018.

[28] M. Weyrich, ‘Towards future Automation Systems in Manufacturing’, presented at the International Conference SIMULATION 2018, Kiev, Ukraine, 2018.

[29] J. Desling, *IoT automation: Arrowhead framework*, CRC Press, Taylor & Francis Group, 2017.

[30] IEC 61131-1, ‘Speicherprogrammierbare Steuerungen Teil 1: Allgemeine Informationen’, Mar. 2004.

[31] VDI-Statusreport, ‘Testen vernetzter Systeme für die Industrie 4.0’, Apr. 2018.

[32] A. Zeller and M. Weyrich, ‘Herausforderung Test verteilter Systeme - Wie Industrie 4.0 das Testen verändert’, *atp edition - Automatisierungstechnische Praxis, no. 10*, 2015.

[33] A. Zeller and M. Weyrich, ‘Challenges for functional testing of reconfigurable production systems’, in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016.

[34] R. Lauber and P. Göhner, *Prozessautomatisierung 1*, 3. Auflage, Springer, 1999.

[35] Acatec, ‘Umsetzungsempfehlung für das Zukunftsprojekt Industrie 4.0’, 2013.

[36] Y. Koren *et al.*, ‘Reconfigurable Manufacturing Systems’, *CIRP Annals*, vol. 48, no. 2, pp. 527–540, Jan. 1999.

[37] G. Reinhart *et al.*, ‘Cyber-Physische Produktionssysteme’, *Wt-Online 2-2013 Sonderheft Industrie 40*, 2013.

[38] Bundesministerium für Wirtschaft und Energie, Ed., ‘Industrie 4.0 und Digitale Wirtschaft - Impulse für Wachstum, Beschäftigung und Innovation’, Apr. 2015.

[39] J. Schlick, P. Stephan, M. Loskyll, and D. Lappe, ‘Industrie 4.0 in der praktischen Anwendung’, in *Industrie 4.0 in Produktion, Automatisierung und Logistik*, pp. 57–84, T. Bauernhansl, M. ten Hompel, and B. Vogel-Heuser, Eds. Springer Fachmedien Wiesbaden, 2014.

[40] VDI/VDE GMA, ‘Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation’, 2013.

[41] A. Roth, Ed., *Einführung und Umsetzung von Industrie 4.0*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[42] A. Fay, B. Vogel-Heuser, T. Frank, K. Eckert, T. Hadlich, and C. Diedrich, ‘Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns’, *Journal of Systems and Software.*, vol. 101, pp. 221–235, Mar. 2015.

[43] VDI/VDE 2653 and VDI/VDE 2653 Blatt 1, ‘Agentensysteme in der Automatisierungstechnik Grundlagen’, May 2018.

[44] J.-P. Schmidt, A. Zeller, and M. Weyrich, ‘Modellgetriebene Entwicklung serviceorientierter Anlagensteuerungen’, *at – Automatisierungstechnik*, vol. 65, no. 1, Jan. 2017.

[45] M. Gharbi, A. Koschel, A. Rausch, and G. Starke, *Basiswissen für Softwarearchitekten: Aus- und Weiterbildung nach ISTQB-Standard zum Certified Professional for Software Architecture - Foundation Level*, 3., überarbeitete und aktualisierte Auflage. Heidelberg: dpunkt.verlag, 2018.

[46] N. M. Josuttis, *SOA in der Praxis: System-Design für verteilte Geschäftsprozesse*, 1. Auflage, Heidelberg: dpunkt.verlag, 2009.

[47] J. M. Mendes, P. Leitao, A. W. Colombo, and F. Restivo, ‘Service-oriented control architecture for reconfigurable production systems’, *6th IEEE International Conference on Industrial Informatics*, 2008.

[48] ANSI/ISA-95.00.01, ‘Enterprise-Control System Integration Part 1: Models and Terminology’, 2000.

[49] H. Bloch *et al.*, ‘Model-based engineering of CPPS in the process industries’, in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, pp. 1153–1159, 2017.

[50] I. Melzer and S. Eberhard, *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*, 4. Aufl. Heidelberg: Spektrum Akad. Verl, 2010.

[51] M. Becker and S. Klingner, ‘Formale Modellierung von Komponenten und Abhängigkeiten zur Konfiguration von Product-Service Systems’, in *Dienstleistungsmodellierung 2012*, O. Thomas and M. Nüttgens, Eds. Wiesbaden: Springer Fachmedien Wiesbaden, 2013.

[52] S. K. Panda, T. Schröder, L. Wisniewski, and C. Diedrich, ‘Plug&Produce Integration of Components into OPC UA based data-space’, in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018.

[53] C. Diedrich *et al.*, ‘Semantic interoperability for asset communication within smart factories’, *22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2017.

[54] OASIS Standard, ‘Reference Model for Service Oriented Architecture 1.0’, Oct. 2006.

[55] VDI / VDE-GMA, ‘Status Report Industrie 4.0 Service Architecture Basic concepts for interoperability’, Nov. 2016.

[56] M. Hoffmann, C. Büscher, T. Meisen, and S. Jeschke, ‘OPC UA-basierte Multi-Agenten-Systeme’, *atp edition - Automatisierungstechnische Praxis*, vol. 58, p. 46, Jul. 2016.

[57] P. Leitão, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo, ‘Smart Agents in Industrial Cyber–Physical Systems’, *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1086–1101, May 2016.

[58] P. Marks, T. Müller, D. Vögeli, T. Jung, N. Jazdi, and M. Weyrich, ‘Agent Design Patterns for Assistance Systems in various Domains – a Survey’, *IEEE 14th International Conference on Automation Science and Engineering (CASE)*, Aug. 2018.

[59] D. Pantförder, F. Mayer, C. Diedrich, P. Göhner, M. Weyrich, and B. Vogel-Heuser, ‘Agentenbasierte dynamische Rekonfiguration von vernetzten intelligenten Produktionsanlagen’, in *Handbuch Industrie 4.0 Bd.2*, B. Vogel-Heuser, T. Bauernhansl, and M. ten Hompel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 31–44, 2017.

[60] VDI/VDE-Richtlinie 3682 Blatt 1, ‘Formalisierte Prozessbeschreibungen - Konzept und grafische Darstellung’, May 2015.

[61] H. Stachowiak, *Allgemeine Modelltheorie*. Wien: Springer, 1973.

[62] M. Winter, T. Roßner, C. Brandes, and H. Götz, *Basiswissen Modellbasierter Test: Aus- und Weiterbildung zum ISTQB Foundation Level - Certified Model-Based Tester*, 2. Auflage, Heidelberg: dpunkt.verlag GmbH, 2016.

[63] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Boston: Pearson Education, 2010.

[64] R. Langmann, Ed., *Taschenbuch der Automatisierung*, 3., neu Bearbeitete Auflage. München: Fachbuchverlag Leipzig im Carl Hanser Verlag, 2017.

[65] R. Lauber, ‘Prozessautomatisierung 2’, Springer-Verlag, 1999.

[66] R. Drath, *Datenaustausch in der Anlagenplanung mit AutomationML: Integration von CAEX, PLCopen XML und COLLADA*. Springer-Verlag, 2009.

[67] P. Hehenberger, B. Vogel-Heuser, D. Bradley, B. Eynard, T. Tomiyama, and S. Achiche, ‘Design, modelling, simulation and integration of cyber physical systems: Methods and applications’, *Computers in Industry*, vol. 82, pp. 273–289, Oct. 2016.

[68] OMG Systems Modeling Group, ‘SysML 1.4 Spezifikation’, Sep. 2015.

[69] M. Broy and K. Stolen, *Specification and Development of Interactive Systems Focus on Streams, Interfaces, and Refinement*. New York, NY: Springer New York, 2001.

[70] N. Diernhofer, ‘IT-Dienstmodellierung’, Universität München, 2007.

[71] J. Lunze, *Ereignisdiskrete Systeme, Modellierung und Analyse dynamischer Systeme mit Automaten, Markovketten und Petrinetzen*. Berlin, Boston: De Gruyter, 2009.

[72] H.-J. Zander, *Steuerung ereignisdiskreter Prozesse*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015.

[73] R. Zurawski and M. Zhou, ‘Petri nets and industrial applications: A tutorial’, *IEEE Trans. Ind. Electron.*, vol. 41, no. 6, pp. 567–583, Dec. 1994.

[74] ISO/IEC 15909-1, ‘Systems and software engineering -- High-level Petri nets -- Part 1: Concepts, definitions and graphical notation’, Dec. 2004.

[75] K. Jensen, *Coloured Petri Nets*. Springer-Verlag Berlin Heidelberg, 2014.

[76] ISO/IEC 15909-2, ‘Systems and software engineering -- High-level Petri nets -- Part 2: Transfer format’, Feb. 2011.

[77] R. König, ‘Petri Nets and their Application for a Standardizable Design of Switching Circuits’, IFAC Proceedings Volumes, vol. 10, no. 2, pp. 112–119, Mar. 1977.

[78] D. Abel and K. Lemmer, *Theorie ereignisdiskreter Systeme: Tutorium des GMA-Fachausschusses 1.8" Methoden der Steuerungstechnik*. München: Oldenbourg Wissenschaftsverlag, 1998.

[79] M. Otter, K.-E. Årzén, and I. Dressler, ‘StateGraph – A Modelica Library for Hierarchical State Machines’, *Proceedings of the 4th International Modelica Conference*, 2005.

[80] M. Rausch and H.-M. Hanisch, ‘Net condition/event systems with multiple condition outputs’, in *INRIA/IEEE Emerging Technologies and Factory Automation (ETFA),* 1995.

[81] W. M. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf, ‘Multiparty contracts: Agreeing and implementing interorganizational processes’, *The Computer Journal*, vol. 53, no. 1, pp. 90–106, 2010.

[82] DIN EN 61131-3:2013, ‘Speicherprogrammierbare Steuerungen – Teil 3: Programmiersprachen’, Jun. 2014.

[83] OMG Object Management Group, ‘Business Process Model and Notation (BPMN), Version 2.0’, 2011.

[84] OASIS Standard, ‘Web Services Business Process Execution Language Version 2.0’, Apr. 2007.

[85] Software AG, ‘ARIS Method Version 9.8 SR6’, Oct. 2016.

[86] W. Reisig, K. Schmidt, and C. Stahl, ‘Kommunizierende Workflow-Services modellieren und analysieren’, *Informatik – Forschung und Entwicklung*, vol. 20, no. 1–2, pp. 90–101, Oct. 2005.

[87] S. Buchwald and T. Bauer, ‘Modellierung von Service-Aufrufbeziehungen zwischen prozessorientierten Applikationen’, *EMISA Forum*, vol. 30, 2010.

[88] V. Dubinin, V. Vyatkin, and T. Pfeiffer, ‘Engineering of Validatable Automation Systems Based on an Extension of UML Combined With Function Blocks of IEC 61499’, in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005.

[89] DIN 25424-1, ‘Fehlerbaumanalyse - Methode und Bildzeichen’. Beuth Verlag GmbH, Sep. 1981.

[90] DIN 25419, ‘Ereignisablaufanalyse - Verfahren, graphische Symbole und Auswertung’, Nov. 1985.

[91] J. Ladiges *et al.*, ‘Entwurf, Modellierung und Verifikation von Serviceabhängigkeiten in Prozessmodulen’, *at – Automatisierungstechnik*, vol. 66, no. 5, pp. 418–437, May 2018.

[92] B. Li, ‘Managing Dependencies in Component-Based Systems Based on Matrix Model’, *Proceedings NetObjectDays*, 2003.

[93] P. Liggesmeyer, *Software-Qualität: Testen, Analysieren und Verifizieren von Software*, 2. Auflage. Heidelberg: Spektrum Akademischer Verlag, 2009.

[94] DIN EN ISO 9000 - 2015-11, ‘Qualitätsmanagementsysteme - Grundlagen und Begriffe’, Nov. 2015.

[95] ISO/IEC/IEEE 24765, ‘Systems and software engineering -- Vocabulary’, Sep. 2017.

[96] SQS, ‘ISTQB Certified Tester - Foundation Level: Grundlagen des Softwaretestens, Q30100, Version 3.7’, 2017.

[97] M. Khalgui, ‘NCES-based modelling and CTL-based verification of reconfigurable embedded control systems’, *Computers in Industry*, vol. 61, no. 3, pp. 198–212, Apr. 2010.

[98] P. Duan, Y. Zhou, X. Gong, and B. Li, ‘A Systematic Mapping Study on the Verification of Cyber-Physical Systems’, *IEEE Access*, vol. 6, pp. 59043–59064, 2018.

[99] S. Magnus, J. Krause, and C. Diedrich, ‘Automatisierte Modellgenerierung auf Basis formalisierter Anforderungsbeschreibung’, *at – Automatisierungstechnik*, vol. 63, no. 2, Jan. 2015.

[100] ‘UML Testing Profile (UTP)’. [Online]. Available: http://utp.omg.org/. [Accessed: 16-Nov-2018].

[101] S. Rösch, S. Ulewicz, J. Provost, and B. 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*, vol. 08, no. 09, pp. 499–519, 2015.

[102] S. Magnus, T. Ruß, J. Krause, and C. Diedrich, ‘Modellsynthese für die Testfallgenerierung sowie Testdurchführung unter Nutzung von Methoden zur Netzwerkanalyse’, *at – Automatisierungstechnik*, vol. 65, no. 1, Jan. 2017.

[103] M. Reider, S. Magnus, and J. Krause, ‘Feature-Based Testing by Using Model Synthesis, Test Generation and Parameterizable Test Prioritization’, in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2018.

[104] J. Krause and C. Diedrich, *Testfallgenerierung aus modellbasierten Systemspezifikationen auf der Basis von Petrinetzentfaltungen*, 1. Aufl. Aachen: Shaker, 2012.

[105] J. Provost, J.-M. Roussel, and J.-M. Faure, ‘Translating Grafcet specifications into Mealy machines for conformance test purposes’, *Control Engineering Practice*, vol. 19, no. 9, pp. 947–957, Sep. 2011.

[106] R. Hametner, B. Kormann, B. Vogel-Heuser, D. Winkler, and A. Zoitl, ‘Test case generation approach for industrial automation systems’, in *5th International Conference on Automation, Robotics and Applications (ICARA)*, 2011.

[107] T. Hussain and G. Frey, ‘UML-based Development Process for IEC 61499 with Automatic Test-case Generation’, in *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2006.

[108] S. Rösch, ‘Model-based testing of fault scenarios in production automation’, Technische Universität München, sierke Verlag, 2016.

[109] B. Kormann and B. Vogel-Heuser, ‘Automated test case generation approach for PLC control software exception handling using fault injection’, in *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, 2011.

[110] S. Magnus, T. Schörder, J. Krause, S. Süß, A. Strahilov, and S. Gulan, ‘Testautomatisierung in der virtuellen Inbetriebnahme’, presented at the AUTOMATION 2015, Baden Baden, 2015.

[111] G. Frey and L. Litz, ‘Formal methods in PLC programming’, presented at the 2000 IEEE International Conference on Systems, Man and Cybernetics, vol. 4, pp. 2431–2436, 2000.

[112] V. Vyatkin, ‘Modelling and verification of discrete control systems’, 2007.

[113] S. Preuße, C. Gerber, and H.-M. Hanisch, ‘Virtual Start-Up of Plants using Formal Methods’, *Journal International Journal of Computer Applications in Technology*, vol. 42, Jan. 2011.

[114] E. Wassermann and A. Fay, ‘Test- und Verifikationsverfahren für Agentensysteme – Status Quo und weitere Herausforderungen’, *at - Automatisierungstechnik*, vol. 65, no. 11, Jan. 2017.

[115] J. O. Blech, P. Lindgren, D. Pereira, V. Vyatkin, and A. Zoitl, ‘A Comparison of Formal Verification Approaches for IEC 61499’, presented at the Emerging Technologies and Factory Automation (ETFA), Berlin, 2016.

[116] V. Vyatkin and H.-M. Hanisch, ‘Verification of distributed control systems in intelligent manufacturing’, *Journal of Intelligent Manufacturing*, vol. 14, no. 1, pp. 123–136, 2003.

[117] C. Gerber, I. Ivanova-Vasileva, and H.-M. Hanisch, ‘Formal modelling of IEC 61499 function blocks with integer-valued data types’, *Control and Cybernetics*, vol. 39, 2010.

[118] V. Vyatkin and H. Hanisch, ‘Development of adequate formalisms for verification of IEC 1499 distributed applications’, in *SICE 2000. Proceedings of the 39th SICE Annual Conference*, 2000.

[119] I. Ivanova-Vasileva, C. Gerber, and H.-M. Hanisch, ‘Basics of Modelling IEC 61499 Function Blocks with Integer-Valued Data Types’, *IFAC Proc. Vol.*, vol. 41, no. 3, pp. 169–174, Jan. 2008.

[120] H.-M. Hanisch and V. Vyatkin, ‘Modeling and Verification of Distributed Control Systems’, presented at the Design, Analysis, and Simulation of Distributed Systems Symposium, 2005.

[121] J. Zhang, ‘Modeling and verification of reconfigurable discrete event control systems’, Xidian University, 2015.

[122] J. Zhang *et al.*, ‘Modeling and Verification of Reconfigurable and Energy-Efficient Manufacturing Systems’, *Discrete Dynamics in Nature and Society*, 2015.

[123] C. Legat *et al.*, ‘Interface Behavior Modeling for Automatic Verification of Industrial Automation Systems’ Functional Conformance’, *at - Automatisierungstechnik.*, vol. 62, no. 11, Jan. 2014.

[124] M. Lochau, S. Mennicke, H. Baller, and L. Ribbeck, ‘Incremental model checking of delta-oriented software product lines’, *Journal of Logical and Algebraic Methods in Programming*, vol. 85, no. 1, pp. 245–267, Jan. 2016.

[125] J. Ladiges, C. Haubeck, A. Fay, and W. Lamersdorf, ‘Evolution Management of Production Facilities by Semi-Automated Requirement Verification’, *at – Automatisierungstechnik*, vol. 62, no. 11, Jan. 2014.

[126] A. Zeller and M. Weyrich, ‘Component based Verification of Distributed Automation Systems based on Model Composition’, *Procedia CIRP*, vol. 72, pp. 352–356, Jan. 2018.

[127] A. Zeller and M. Weyrich, ‘Composition of Modular Models for Verification of Distributed Automation Systems’, *Procedia Manufacturing*, vol. 17, pp. 870–877, Jan. 2018.

[128] A. Zeller, N. Jazdi, and M. Weyrich, ‘Verifikation verteilter Automatisierungssysteme auf Basis einer Modellkomposition’, at – *Automatisierungstechnik*, vol. 66, no. 6, pp. 456–470, 2018.

[129] G. Frey and L. Litz, ‘Verification and validation of control algorithms by coupling of interpreted petri nets’, in *IEEE International Conference on Systems, Man, and Cybernetics*, 1998.

[130] A. Eisener, ‘Realisierung eines Algorithmus zur effizienten Absicherung serviceorientierter Automatisierungssysteme (Masterarbeit 2910)’, *Institut für Automatisierungstechnik und Softwaresysteme Universität Stuttgart*, Sep. 2017.

[131] K. Land, ‘Realisierung eines Testsystems für verteilte Automatisierungssysteme (Masterarbeit 2923)’, *Institut für Automatisierungstechnik und Softwaresysteme Universität Stuttgart*, Nov. 2017.

[132] P. Schleissinger, ‘Aufbau einer Testkomponente zur modellbasierten Absicherung verteilter Automatisierungssysteme (Masterarbeit 2958)’, *Institut für Automatisierungstechnik und Softwaresysteme Universität Stuttgart*, Apr. 2018.

[133] ‘Prosys OPC UA Java SDK - Prosys OPC’. [Online]. Available: https://www.prosysopc.com/products/opc-ua-java-sdk/. [Accessed: 15-Mar-2019].

[134] ‘Homepage of ITS-tools | ITS Tools’. [Online]. Available: https://lip6.github.io/ITSTools-web/. [Accessed: 30-Oct-2018].

[135] Joy-IT, ‘Touch-Display 10,1 Zoll’. [Online]. Available: http://anleitung.joy-it.net/wp-content/uploads/2016/12/RB-LCD10-Datenblatt.pdf. [Accessed: 15-Mar-2019].

[136] T. Pfeiffer, ‘Realisierung eines verteilten IT-Systems auf Basis von OPC UA’ (Masterarbeit), *Institut für Automatisierungstechnik und Softwaresysteme Universität Stuttgart*, Jan. 2017.

[137] W. M. Rösch, ‘Realisierung einer service-orientierten Steuerung für ein modulares Produktionssystem (Masterarbeit 2933)’, *Institut für Automatisierungstechnik und Softwaresysteme Universität Stuttgart*, Dec. 2017.

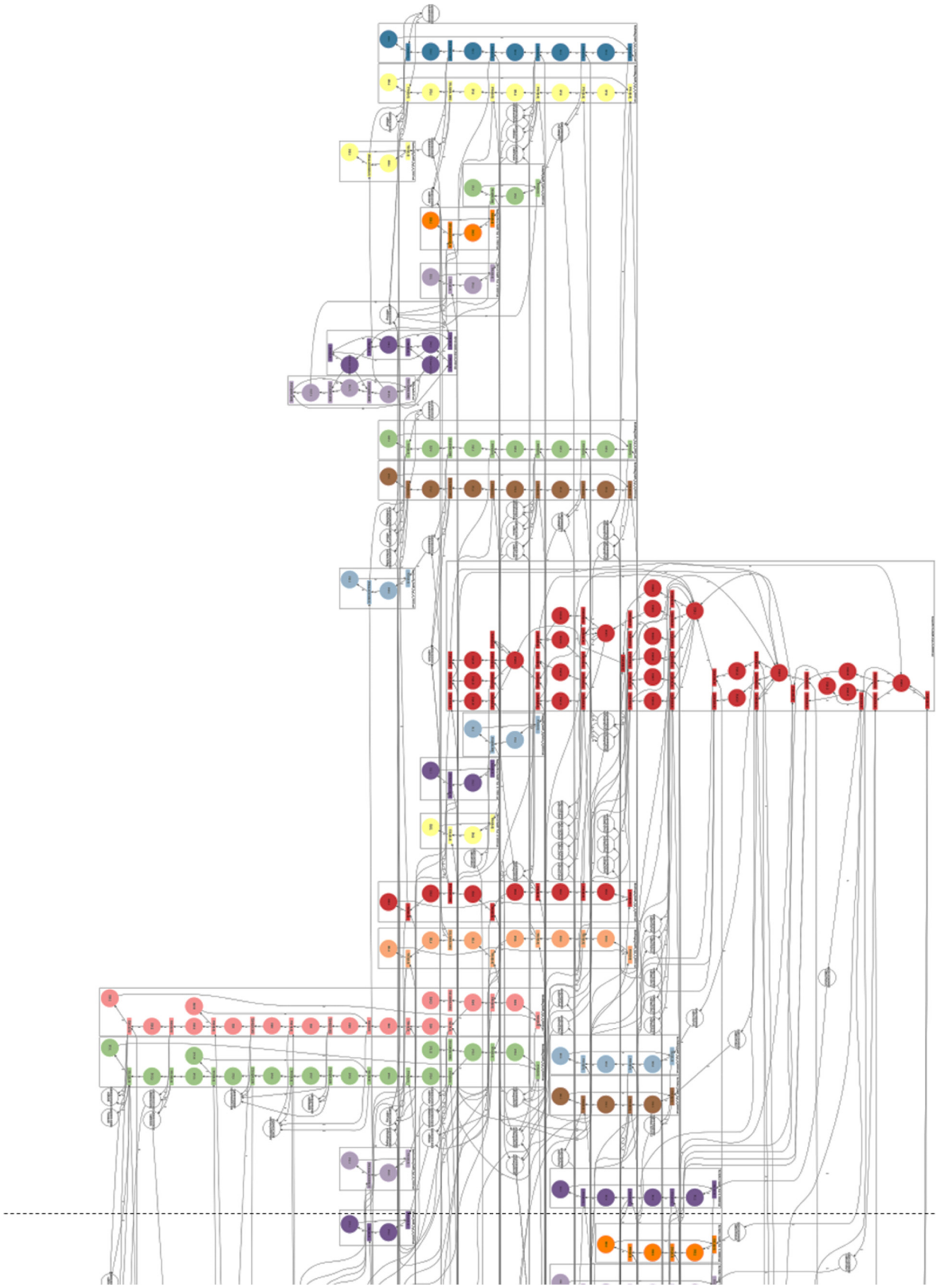
[138] M. Alves Pais Alvarenga, ‘Entwurf und Realisierung einer Schnittstelle zur Rekonfiguration eines OPC-UA Netzwerks (Masterarbeit 2955)‘, *Institut für Automatisierungstechnik und Softwaresysteme, Universität Stuttgart*, Mar. 2018.

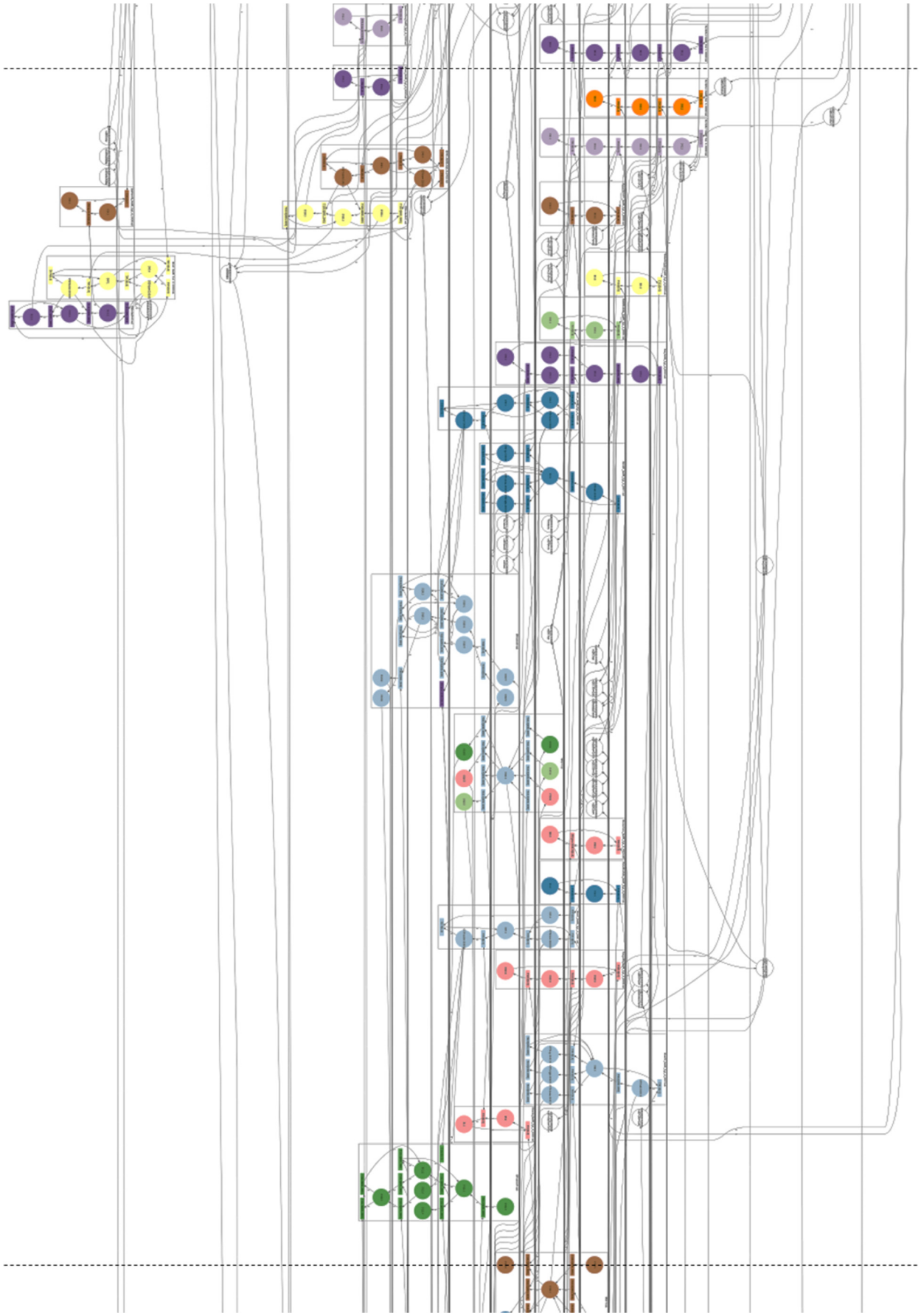
[139] T. Abele, ‘Entwurf und Integration einer Topologie in ein OPC-UA-basiertes Automatisierungssystem (Masterarbeit 2965)’, *Institut für Automatisierungstechnik und Softwaresysteme Universität Stuttgart*, Apr. 2018.

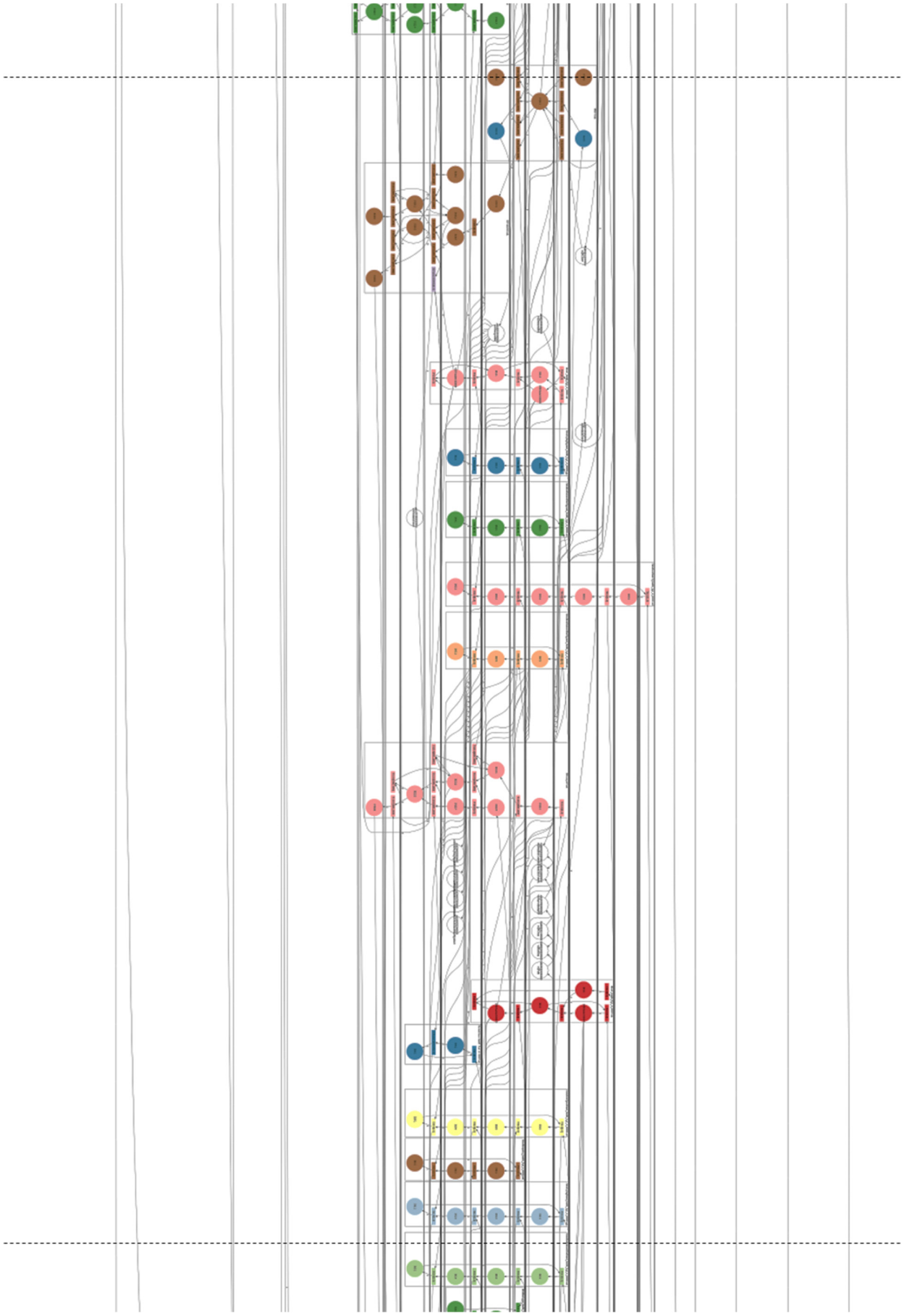
[140] L. Balzer, A. Frey, and P. Nenniger, ‘Was ist und wie funktioniert Evaluation?’, *Zeitschrift zu Theorie und Praxis erziehungswissenschaftlicher Forschung*, 1999.

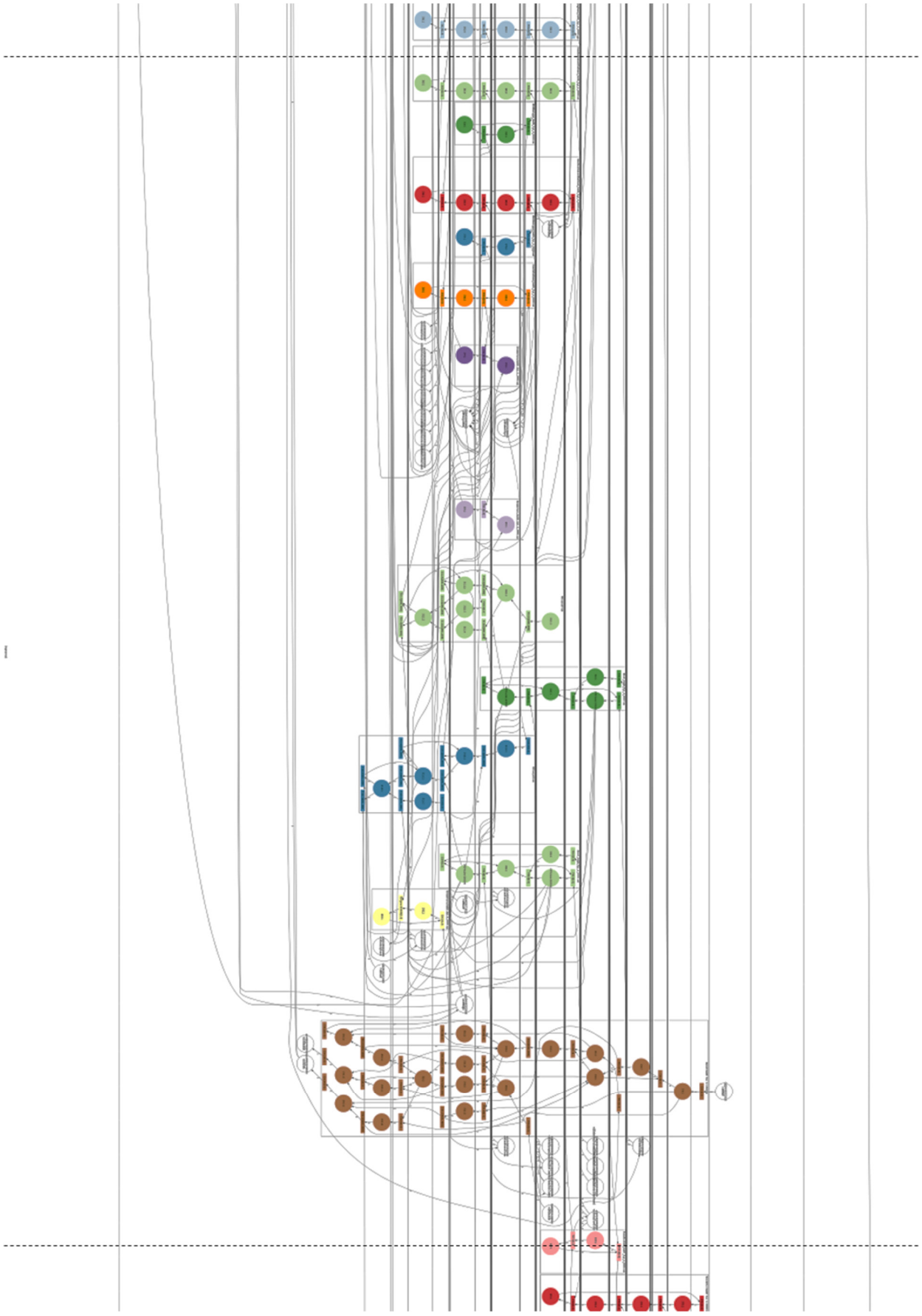
Anhang

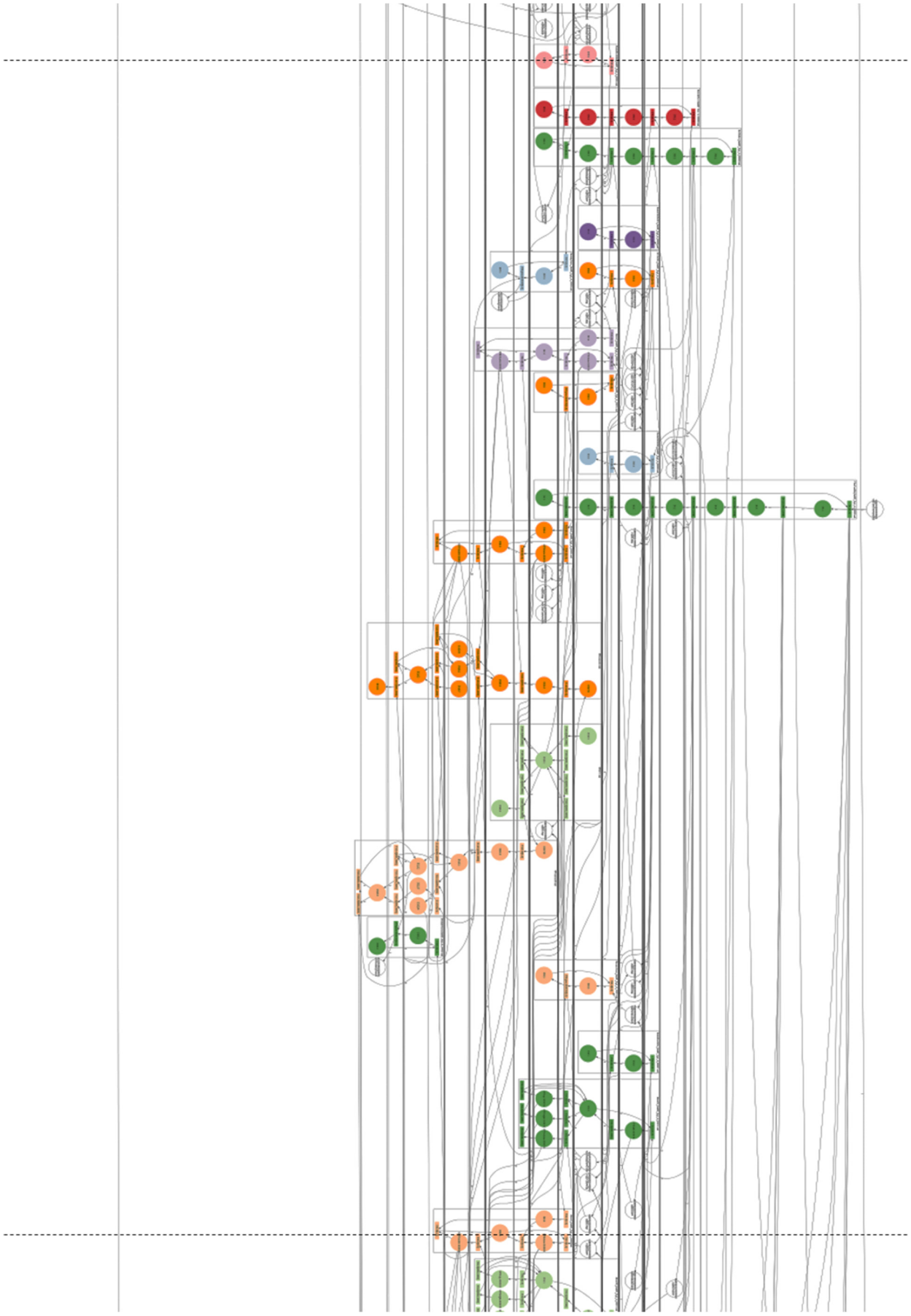
Darstellung des komponierten Petri-Netzes des gesamten verteilten Steuerungssystems

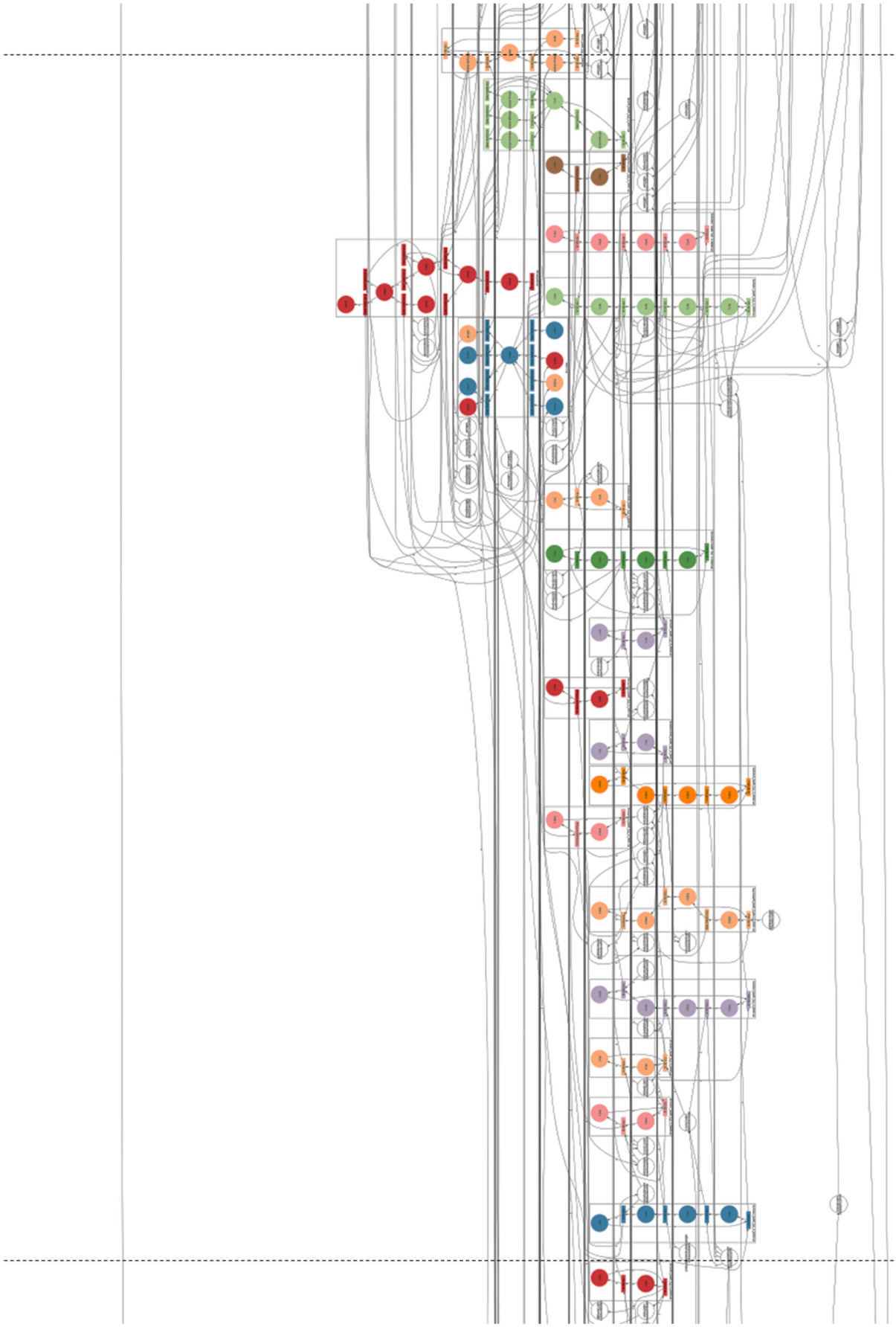


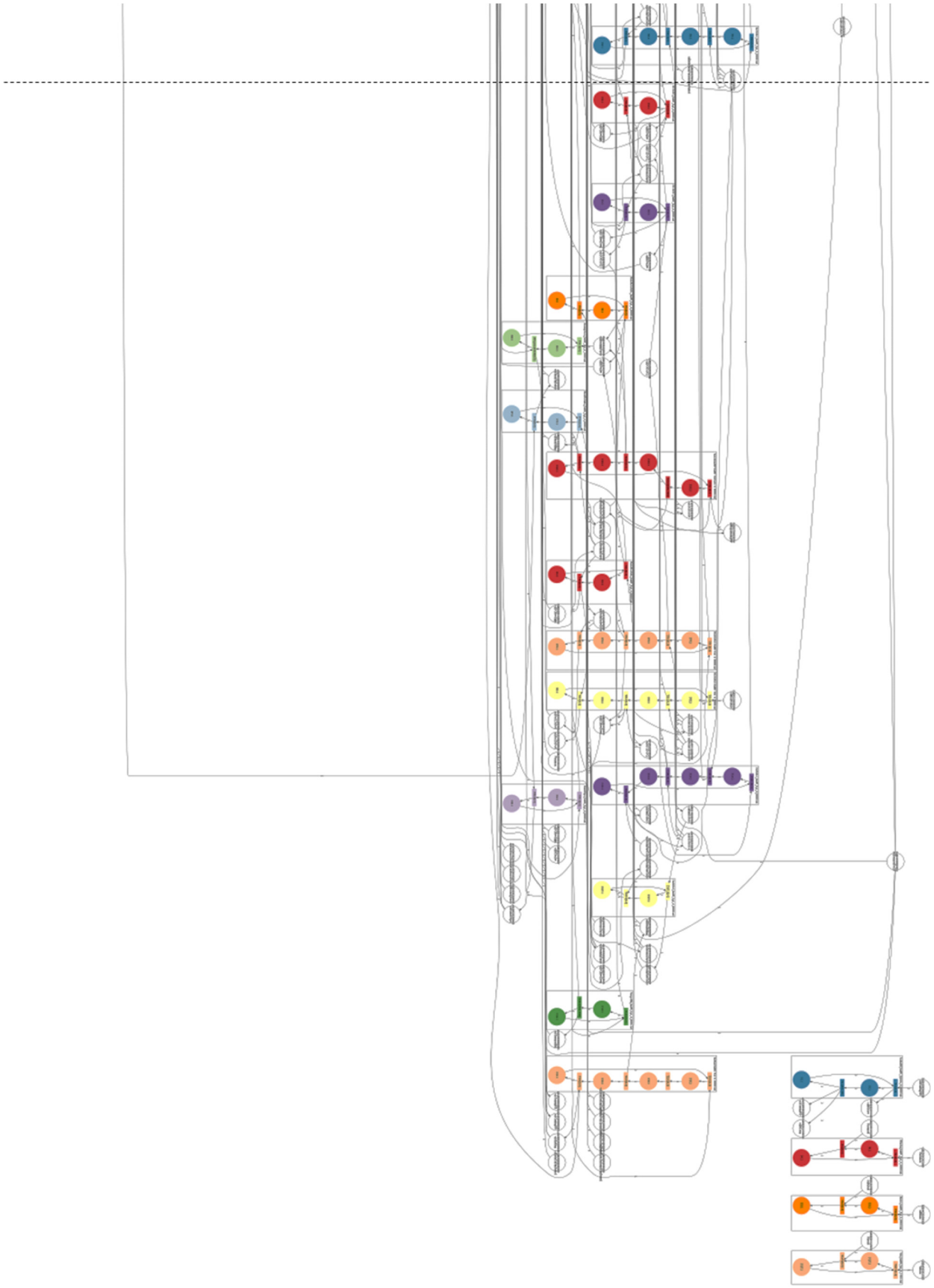




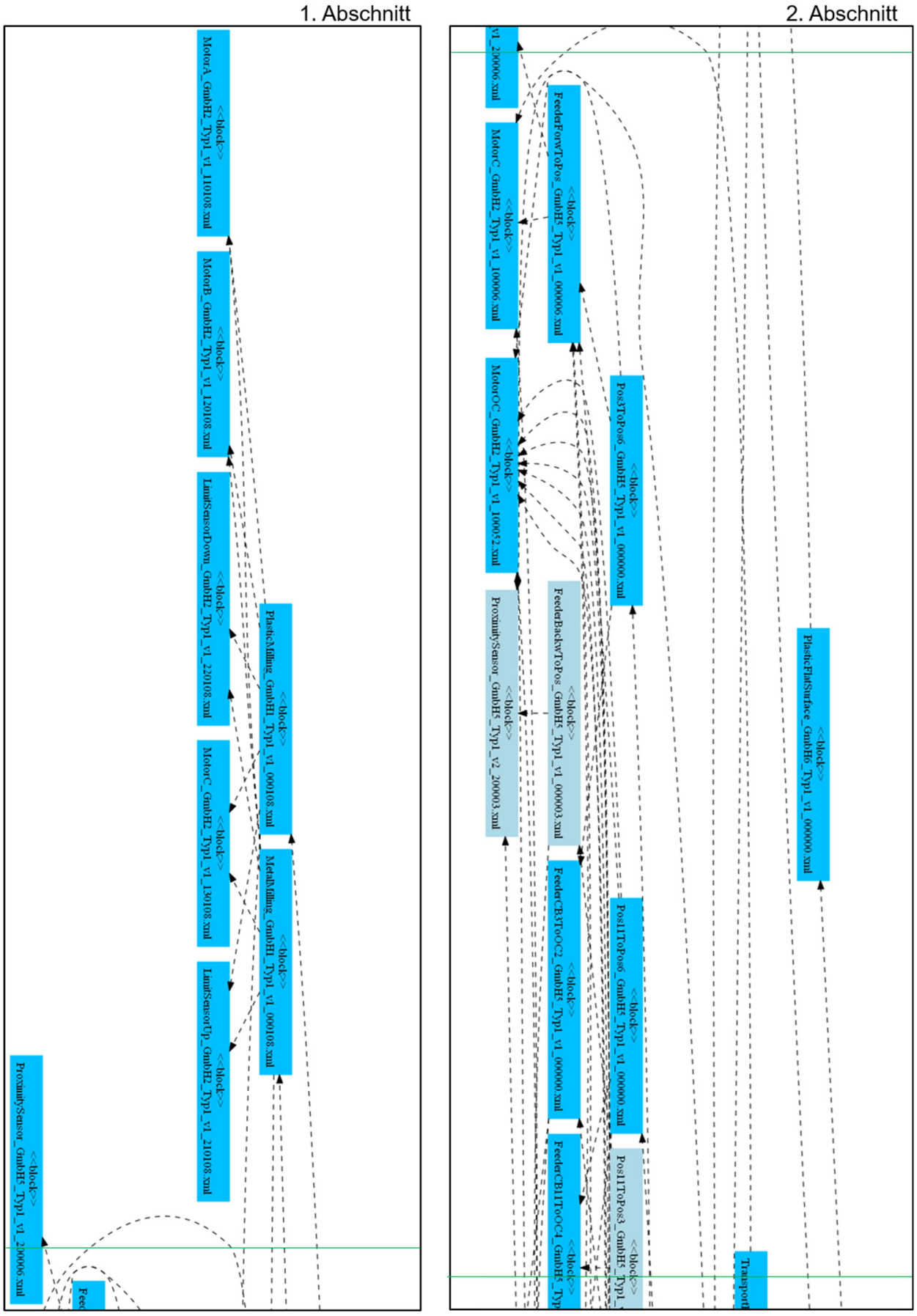


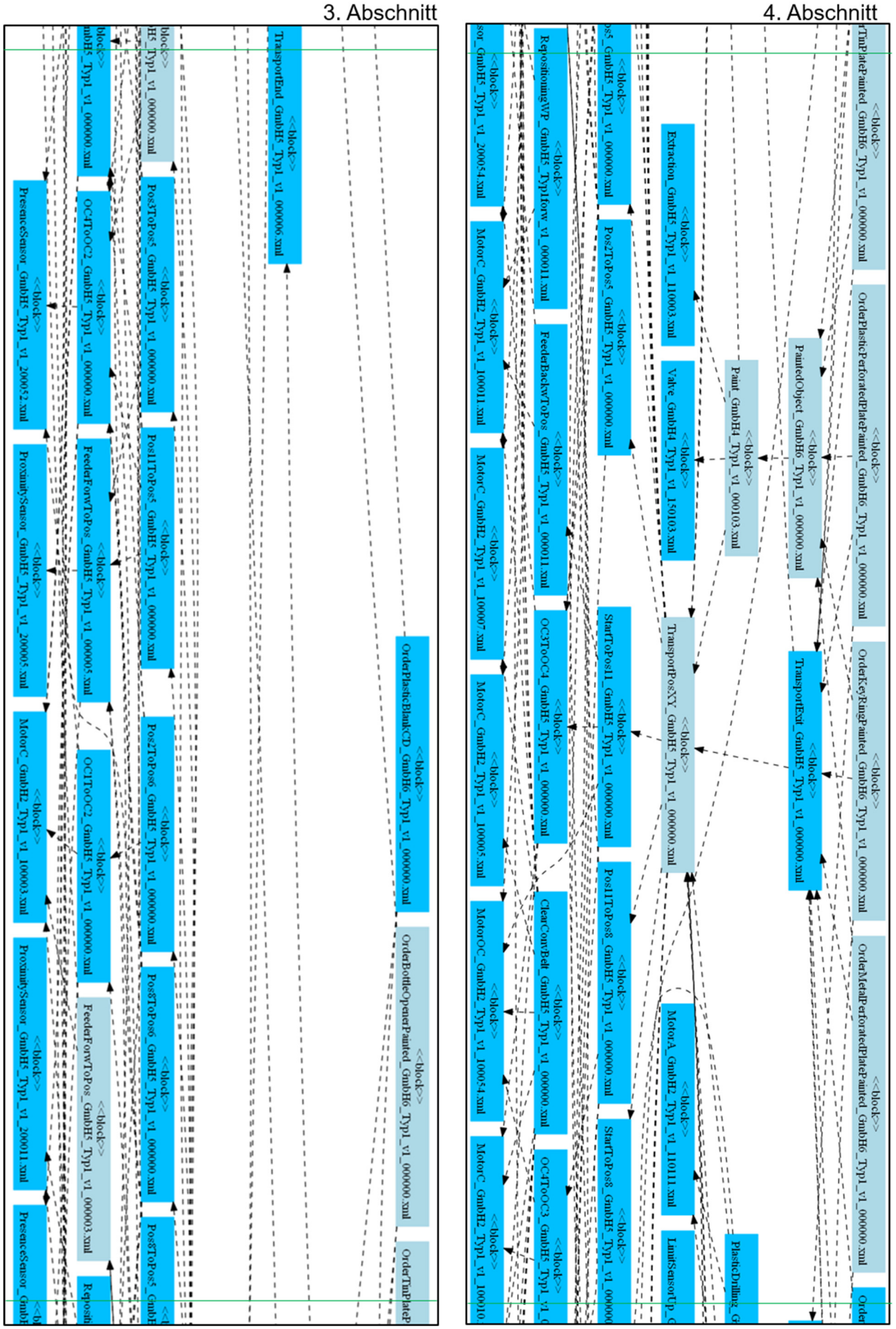


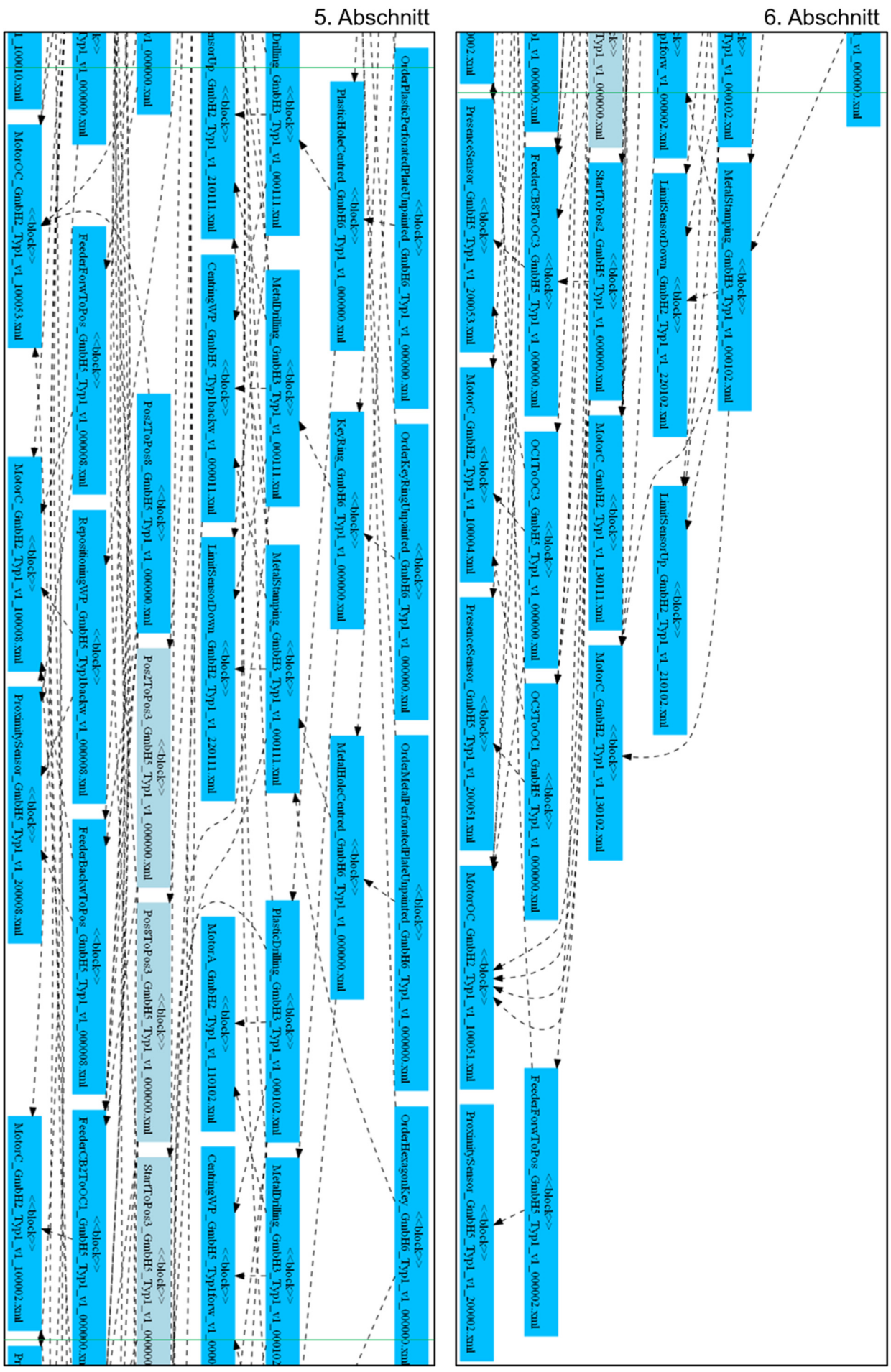




Darstellung der Abhängigkeiten innerhalb des verteilten Steuerungssystems




3. Abschnitt
4. Abschnitt


6. Abschnitt
5. Abschnitt