Andreas Zeller, Dr. Nasser Jazdi, Prof. Michael Weyrich

Verifikation verteilter Automatisierungssysteme auf Basis einer Modellkomposition

Verification of distributed automation systems based on a model composition

Zusammenfassung:

Änderungen der Funktionalität von Automatisierungssystemen werden zunehmend an der Software durchgeführt. Insbesondere bei verteilten Automatisierungssystemen lassen sich die Auswirkungen von Software-Änderungen oftmals nur schwer abschätzen. Dies stellt insbesondere Anlagenbetreiber vor große Herausforderungen bei der Absicherung von Funktions-Änderungen in der Betriebsphase. Dieser Beitrag beschreibt einen modellbasierten Ansatz, mit dem Auswirkungen von Software-Änderungen erkannt werden können und das betroffene Teilsystem automatisiert über modellbasierte Verifikationsverfahren abgesichert werden kann. Das präsentierte Konzept beruht auf der Komposition eines Systemmodells aus diskreten Verhaltensmodellen der Komponenten des Automatisierungssystems. Abschließend wird die das Konzept anhand unterschiedlicher Änderungsszenarien evaluiert.

Abstract:

Changes of the functionality of an industrial automation system are increasingly achieved by software modifications. The effects of software modifications are often difficult to estimate, especially in distributed automation systems. Mainly, this will challenge plant operators who have to safeguard their automation systems after functionality changes were executed. This contribution describes a model based approach which detects the impact of software modifications and, subsequently, allows to automatically safeguard affected subsystems with the help of model based verification methods. The presented concept is based on the generation of a system model which is composed by models of the components of the automation system.

1 Einleitung

Sinkende Kosten für digitale Komponenten und ausgereifte Entwicklungstools und –verfahren führen dazu, dass zunehmend Funktionalität in Software realisiert wird. Dies ermöglicht die Umsetzung komplexer Automatisierungsaufgaben und, durch die vergleichsweise einfache Änderbarkeit von Software, eine hohe Flexibilität von Automatisierungssystemen [1]. So bilden Softwaresysteme die Basis für neuartige Geschäftsmodelle für Anlagenbetreiber, wie die Herstellung kundenspezifischer Produkte auf Basis einer flexiblen, wandlungsfähigen Produktion [2]. Die Bedienung dieser Flexibilitätsanforderungen erfordert einen elementaren Wandel der Entwurfsmuster von Automatisierungssystemen.

Zur Umsetzung einer flexiblen Produktion ist ein durchgängiger Informationsfluss zwischen den Komponenten des Automatisierungssystems unerlässlich [2] [3]. Dies führt dazu, dass besonders Komponenten der Feldebene komplexere Kommunikationsmechanismen beherrschen müssen, um Daten an verschiedene Adressaten senden sowie Parametrisierungsdaten und Software-Updates empfangen zu können. Über Software-Updates werden Komponenten automatisierter Systeme in der Lage sein neue Steuerungsaufgaben zu übernehmen. Zur Gewährleistung der Wandelbarkeit von Automatisierungssystemen werden dezentral gesteuerte Automatisierungslösungen benötigt [3] [4]. So verteilen sich beispielsweise die Fähigkeiten der Automatisierungssysteme auf viele Komponenten. Dabei soll es möglich sein, Komponenten ad-hoc in Automatisierungssysteme zu integrieren [2]. Agentensysteme sowie serviceorientierte Architekturen könnten aufgrund der losen Kopplung zwischen den Komponenten diese Fähigkeiten bieten [5]. Dies kann eine leichte Änderbarkeit der Steuerungslogik von Komponenten und der Struktur des Automatisierungssystems ermöglichen.

Die Kopplung vieler Komponenten und der durchgängige Informationsfluss führen aber auch dazu, dass die Abhängigkeiten zwischen den Komponenten schwierig nachzuvollziehen sind. Diese Abhängigkeiten sind dabei nicht statisch, sondern verändern sich durch Ad-Hoc-Vernetzung und Software-Änderungen über den Lebenszyklus des Automatisierungssystems [6]. Um Auswirkungen von Funktionsänderungen über Software-Änderungen analysieren zu können, ist die Kenntnis der informationstechnischen Abhängigkeiten zwischen den Komponenten notwendig. Zur Analyse der Auswirkungen eignen sich Systemmodelle, welche die durch die Abarbeitung einer Automatisierungsaufgabe entstehenden Abhängigkeiten zwischen den Komponenten darstellen. Die Bereitstellung eines solchen Systemmodells ist problematisch, da ein Automatisierungssystem meist aus Komponenten unterschiedlicher Hersteller besteht. Dies führt dazu, dass die Kenntnis über die innere Struktur und Abhängigkeiten aller Komponenten des Automatisierungssystems kaum beherrschbar ist. Diese Nichtbeherrschbarkeit stellt die Absicherung von rekonfigurierbaren Systemen vor große Herausforderungen.

Darüber hinaus führt der hohe Funktionsumfang der Steuerungssoftware zu einem hohen Testaufwand. Da, aufgrund verdeckter, sich ändernder Abhängigkeiten, die Auswirkungen von Software-Änderungen auf andere Komponenten häufig nicht erkennbar sind, existiert oft nicht die Möglichkeit diese einzugrenzen. Modellbasierte Verfahren können aufgrund einer abstrahierten Darstellung des Systems sowie der maschinellen Verarbeitbarkeit bei der Absicherung unterstützen.

So kann ein Systemmodell (Verhaltensmodell des gesamten Automatisierungssystems) beispielsweise der Absicherung komponentenübergreifender Funktionalitäten eines verteilten Automatisierungssystems über modellbasierte Verifikationsverfahren dienen. Des Weiteren lassen sich Auswirkungsanalysen von Software-Änderungen durchführen.

Aufgrund der Wandelbarkeit zukünftiger Automatisierungssysteme ist davon auszugehen, dass sich die Struktur des Automatisierungssystems und damit auch die des Systemmodells ständig ändert. Aufgrund der Dynamik des Systemmodells wird der Aufwand zur Erstellung und Pflege des Systemmodells künftig weiter ansteigen. Eine von den Autoren veröffentliche Umfrage ergab zudem, dass der Testaufwand während der Betriebsphase innerhalb der nächsten 10 Jahre stark zunehmen wird [7]. Dies kann unter anderem auf häufige Änderungen in der Betriebsphase und die verteilte Struktur zukünftiger Automatisierungssysteme zurückgeführt werden. Dabei fehlt es an Konzepten, um Anlagenbetreiber bei der Absicherung von Software-Änderungen während der Betriebsphase zu unterstützen.

In diesem Beitrag wird ein Assistenzsystem zur automatisierten und effizienten Absicherung von Automatisierungssystemen bei Software-Änderungen vorgestellt. Dies soll Anlagenbetreiber bei der Absicherung von Automatisierungssystemen während der Betriebsphase unterstützten. Hierfür wird ein Konzept vorgestellt, welches beschreibt, wie anhand von Komponenten-Modellen automatisiert ein Abhängigkeitsdiagramm aufgebaut werden kann. Anhand einer Auswirkungsanalyse werden durch Software-Änderungen betroffene Systemanforderungen identifiziert. Zur modellbasierten Absicherung der betroffenen Systemanforderungen wird ein Verhaltensmodell aus den Komponenten-Modellen komponiert, die zur Verifikation der betroffenen Systemanforderungen notwendig sind. Die anschließende Verifikation der betroffenen Anforderungen ermöglicht zügig eine Aussage darüber, ob die modellierte Steuerungslogik konform mit den geprüften Systemanforderungen ist.

2 Stand der Technik

Änderungen an Automatisierungssystemen können nach [8] in vier verschiedene Klassen eingeteilt werden. So lässt sich unterschieden, ob Änderungen an der Mechanik; an der Mechanik, Sensorik & Aktorik; der Plattform; oder ausschließlich an der Software vorliegen. Abhängig von der Klasse werden die Komponenten angegeben (Kontext, Plattform, Software, Simulation des technischen Prozesses), für welche eine Anpassung nötig ist, um simulationsbasiert zu testen. So muss bei Software-Änderungen ausschließlich die Steuerungssoftware angepasst werden. Dabei können Auswirkungsanalysen helfen, um die Auswirkungen von Softwareänderungen einzugrenzen und nicht das Gesamtsystem erneut absichern zu müssen [9]. So wird unter anderem bei der Fehlermöglichkeits- und -einflussanalyse (FMEA) das Prinzip der Modellierung von Abhängigkeiten zur Analyse der Auswirkungen von Fehlernzuständen verwendet.

Modelle eignen sich sehr gut zur Strukturierung von Artefakten innerhalb des Testprozesses. Es existieren zahlreiche Ausprägungen des modellbasierten Testens. So können beispielsweise die Testarchitektur, die Testumgebung, die Testdaten, die Testfälle sowie das Verhalten des Testobjekts modelliert werden [10]. Zur Überprüfung, ob das Verhalten eines Testobjekts dessen Anforderungen entspricht, können modellbasierte Verifikationsverfahren verwendet werden. Dabei wird das funktionale Systemmodell, welches die Steuerungslogik des Systems abbildet, gegen formalisierte Anforderungen an das System verifiziert. Modellbasierte Verifikation erlaubt es, vor der Inbetriebnahme zu überprüfen, ob die Steuerungslogik den an sie gestellten Anforderungen genügt. Das Model Checking ist ein modellbasiertes Verifikationsverfahren bei welchem das formale Verhaltensmodell eines Systems gegen dessen formalisierte Systemanforderungen mithilfe eines Algorithmus geprüft wird. Formalisierte Systemanforderungen können in temporaler Logik wie CTL (Computational Tree Logic) und LTL (Linear-Time Temporal Logic) formuliert sein. RCTL [11]

(Reconfigurable Computational Tree Logic) erweitert CTL und adressiert Systeme, welche während der Laufzeit ihr Verhalten ändern. Beim Model Checking stellt die Zustandsraumexplosion bei Systemen mit hohem Parallelisierungsgrad eine Herausforderung dar. So gibt es auch im Bereich der Automatisierungstechnik Konzepte, um die Zustandsraumexplosion mithilfe von Domänenwissen einzugrenzen [12]. Da nicht Verifikationsmethoden und Verifikationstools, sondern die Generierung eines geeigneten Eingangs im Fokus der Veröffentlichung ist, werden diese nicht weiter betrachtet.

2.1 Forschungsbestrebungen zur Absicherung verteilter, änderbarer Automatisierungssysteme

Forschungsbestrebungen im Bereich der modellbasierten Absicherung von IEC61499-basierten Systementwicklungen werden in [13] dargestellt und verglichen. Ausgewählte Ansätze, welche die modellbasierte Absicherung verteilter, änderbarer Automatisierungssysteme betrachten, werden im Folgenden betrachtet. Im Rahmen des FOCUS-Projektes wurde eine Methodik zur formalen Entwicklung verteilter, reaktiver Systeme entwickelt. Dabei kann die Konsistenz der entwickelten Systeme in verschiedenen Entwicklungsstufen, auf verschiedenen Abstraktionsebenen, über formale Verifikationsverfahren überprüft werden. Die Modelle der Komponenten des verteilten Systems kooperieren dabei über syntaktisch und semantisch definierte Schnittstellen [14] [15]. Der Schwerpunkt des FOCUS-Projektes bezieht sich auf das Engineering verteilter Systeme und nicht auf Änderungen während der Betriebsphase.

Im Rahmen des DFG Schwerpunktprogramms 1593 wird unter anderem die Absicherung von sich ändernder Software betrachtet. So beschreibt [16] ein Verfahren zur Verifikation von Schnittstellen-Verhaltensmodellen komponentenbasierter Systeme unter Berücksichtigung von Hardware, Software und Elektrik nach Änderungen. Die verwendeten Verhaltensmodelle beinhalten Material-, Energie-, Physikalische sowie Datenschnittstellen. Zur Verifikation wurde das Framework MoDEMAS entwickelt. In [17] wird vorgestellt, wie Änderungen durch Prozessdaten erkannt werden können und diese auf die zugehörigen Systemanforderungen bezogen werden können um eine stetige Verifikation durchführen zu können. Dabei werden Struktur- und Verhaltensmodelle aus den Prozessdaten abgeleitet, welche über ein Informationsmodell semantisch beschrieben sind. Darüber hinaus wird ein Konzept zur effizienten Verifikation von variantenreicher Software vorgestellt. Dabei werden betroffene Pfade identifiziert und über ein inkrementelles Model Checking abgesichert [18]. Das Schwerpunktprogramm 1593 adressiert mit unterschiedlichen Ansätzen die Evolution von Software über den Softwarelebenszyklus. So werden in [17] auch Änderungen während der Betriebsphase adressiert. Dabei basiert die Verifikation des Systems auf Auswertung von Prozessdaten. Aufgrund der benötigten Prozessdaten ist eine Verifikation vor Inbetriebnahme bei diesem Ansatz nicht möglich.

2.2 Modellierungsarten zur Verifikation verteilter, änderbarer Automatisierungssysteme

Die Modellierungstiefe und die Wahl einer geeigneten Modellierungsart können entscheidend für den Erfolg der modellbasierten Verifikation sein. Daher gibt der folgende Abschnitt einen Überblick über formale Modellierungsarten verteilter Steuerungssysteme, welche sich zur Verifikation eignen. Aufgrund der Menge an Modellierungsarten wird nur auf eine Auswahl eingegangen, welche sich zur Modellierung des Verhaltens verteilter Automatisierungssystem sehr gut eignen:

Modelica erlaubt es neben dem Steuerungssystem das mechanische, elektrische, thermodynamische, hydraulische, pneumatische und thermische Verhalten zu modellieren. Somit ist es möglich, das Automatisierungssystem und den technischen Prozess sehr genau in einem geschlossenen Kreis zu beschreiben [19]. Oftmals werden zur Modellierung von Steuerungssystemen Varianten von zeitbehafteten Automaten verwendet, die in passender Tool-Umgebung, wie beispielsweise UPPAAL, gegen formalisierte Anforderungen verifiziert werden können [20]. In [21] wird ein Konzept vorgestellt, um das Zeitverhalten von vernetzten Automatisierungssystemen zu modellieren. Diese Modelle können zur Simulation oder zur modellbasierten Verifikation verwendet werden.

Einige Modellierungsarten basieren auf klassischen Petri-Netzen, welche beim Engineering von Systemen etabliert sind. So existiert eine Norm (ISO/IEC 15909), welche unter anderem die grafische Notation sowie ein Austauschformat für High-Level-Petri-Netze definiert [22] [23]. NCES (Net Condition / Event Systems) wurden entwickelt, um modulare Steuerungssoftware zu modellieren [24]. Die Modellierung basiert auf typischen Petri-Netzen, welche durch Event- und Condition-Signale erweitert sind, um Schnittstellen zwischen den Komponenten zu realisieren. Da sie nicht gemäß der Spezifikation von typischen Petri-Netzen Stellen mit Transitionen verbinden, sondern Stellen mit Stellen oder Transitionen mit Transitionen, stellen diese Verbindungen eine Sonderform dar. Die Modellierungsart wurde in weiteren Arbeiten zur Modellierung verteilter Steuerung verwendet, erweitert und eine Verifikationsumgebung (VEDA) entwickelt, in welcher sich NCES modellieren, verifizieren und analysieren lassen [25] [26]. Die Komponenten können dabei über definierte Schnittstellen verbunden werden.

Im Bereich der Informatik sind modellbasierte Verifikationsverfahren und verteilte, service-orientierte Architekturen etabliert. „Offene Netze“ sind eine modulare Modellierungsart, welche aus dieser Domäne stammt und die Interaktion zwischen Services über asynchrone Kommunikation beschreibt [27]. Dabei werden Interface-Stellen definiert, welche die Funktionalität eines Services nach außen kapseln. Die Interface-Stellen sind konform mit den Schaltregeln typischer Petri-Netze. Da Schnittstellen zum technischen Prozess in der Informatik meist nicht relevant sind, werden diese nicht berücksichtigt.

Steuerungstechnisch interpretierte Petri-Netze (SIPN) können zur Projektierung von Speicherprogrammierbaren Steuerungen (SPS) verwendet werden [28]. Darüber hinaus ist die Ablaufsprache, welche in Norm IEC 61131-3 [29] standardisiert wurde, an Petri-Netze angelehnt.

Liegt kein Verhaltensmodell vor, kann SPS-Code, welcher in IL (Instruction List, IEC 61131-3) geschrieben ist, in ein formales Modell überführt werden. Zur Überführung der IL in formale Modelle gibt es zahlreiche Ansätze [12] [30] [31].

Für das Konzept relevante Eigenschaften der Modellierungsarten sind in *Tabelle 1* zusammengefasst.

Modellierungsart	Darstellung von Parallelität	Schnittstellen: IT (Informations-technisch), TP (Technischer Prozess)	Spezialisierung - Modellierung von Logik	Zeitl. Verhalten	Verifikation
Modelica	Ja	IT, TP	Gering	Ja	Geeignet
Automaten	Nein	IT, TP	Hoch	Ja	Geeignet
[21]	Ja	IT, TP	Mittel	Ja	Geeignet
NCES	Ja	IT, TP	Mittel	Ja	Geeignet
Offene Netze	Ja	IT	Hoch	Ja	Geeignet
SIPN	Ja	TP	Hoch	Ja	Geeignet

Tabelle 1: Übersicht existierender Modellierungssprachen und Umgebungen

Die Modellierungsartenwurden teils für sehr spezielle Anwendungsfälle und teils für ein breites Anwendungsfeld entwickelt. Eine breite Anwendbarkeit muss dabei über eine Zunahme an Komplexität bei der Modellierung erkauft werden. Modelica ist sehr mächtig und lässt eine höchst präzise Darstellung des technischen Prozesses zu, was sich in der Komplexität der Modellierung niederschlägt. Automaten eignen sich zur Darstellung von parallelen Prozessen nur bedingt. Petri-Netze sind aufgrund der einfachen Darstellung von parallelen Abläufen, der maschinellen Verarbeitbarkeit, der Standardisierung der Darstellung und des Datenformats und der Erweiterbarkeit in der Automatisierungstechnik schon weit verbreitet. NCES, Offene Netze und SIPN basieren auf Petri-Netzen, welche die Parallelitäten verteilter Netzwerke gut darstellen können. Im Gegensatz zu offenen Netzen verfügen NCES über zusätzliche Kantentypen. Diese von typischen Petri-Netzen abweichende Struktur schränkt die Nutzbarkeit verfügbarer Verifikationstools ein. Offene Netze eignen sich sehr gut zur Darstellung asynchroner Kommunikation. Durch den Ursprung in der Informatik verfügen diese aber nicht über Schnittstellen zum technischen Prozess. SIPN verfügen über Schnittstellen zum technischen Prozess, berücksichtigen aber nicht die Kommunikationsschnittstellen zwischen Komponenten des Automatisierungssystems.

3 Konzept zur komponentenbasierten Verifikation verteilter Automatisierungssysteme

Das in Abbildung 1 dargestellte Konzept dient dazu, verteilte Automatisierungssysteme bei Software-Änderungen der Steuerungslogik (Funktionsänderung) einer Komponente (dargestellt als Puzzleteil) möglichst automatisiert und zielgerichtet über modellbasierte Verifikationsverfahren abzusichern. Dabei werden statische Software-Änderungen bei gestopptem System betrachtet. Anlagenbetreiber sollen dadurch bei der Absicherung von Software-Änderungen an ihren Automatisierungssystemen unterstützt werden. Eine Komponente beschreibt einen Steuerungsservice, der eine abgegrenzte Teilfunktionalität (z. B. die Steuerung eines Förderbands) beinhaltet. Dabei ist das Ziel des Konzeptes, einen möglichst maßgeschneiderten Eingang für modellbasierte Verifikationswerkzeuge zu generieren.

Im ersten Schritt wird, ausgehend von einer Software-Änderung, eine Auswirkungsanalyse durchgeführt. Dazu wird anhand semantischer Schnittstellen der Komponenten ein Abhängigkeitsdiagramm aufgebaut. Sobald die von einer Software-Änderung betroffenen Anforderungen identifiziert sind, wird im zweiten Schritt für die Anforderungen jeder betroffenen Komponente ein maßgeschneidertes Verhaltensmodell komponiert. Das komponierte Verhaltensmodell beinhaltet alle Modelle der Komponenten, die zur Absicherung der Anforderungen notwendig sind. Dies bildet die Basis für den letzten Schritt, eine modellbasierte Verifikation der betroffenen Anforderungen gegen das komponierte Verhaltensmodell.

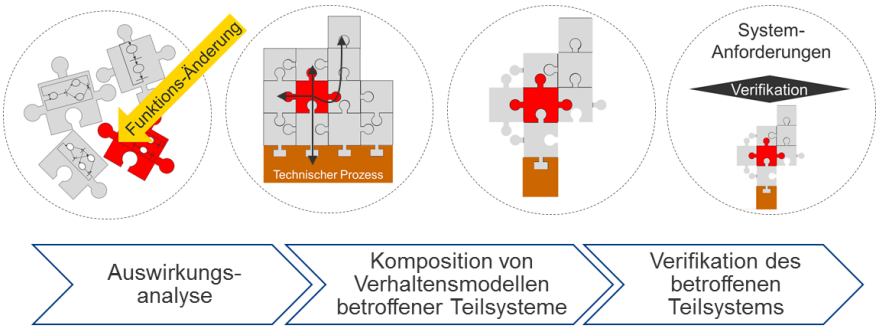

Abbildung 1: Überblick über das Konzept zur komponentenbasierten Verifikation verteilter Automatisierungssysteme

Im folgenden Abschnitt 3.1 wird das für die Illustration des Konzepts verwendete **Szenario** beschrieben. Anschließend wird der komponentenbasierte Modellierungsansatz in Abschnitt 3.2 vorgestellt. Darin wird das Prinzip einer logisch hierarchischen Anforderungsdefinition für serviceorientierte Steuerungen eines Automatisierungssystems vorgestellt. Im darauffolgenden Abschnitt 3.3 werden Anforderungen definiert, die eine **Modellierungsart** erfüllen muss, um für das Konzept geeignet zu sein. Eine geeignete Modellierungsart wird anschließend definiert und Formeln zur Komposition dargelegt. Dies bildet die Grundlage für die **Auswirkungsanalyse** (Abschnitt 3.4), welche im nächsten Abschnitt erläutert wird. Als Ergebnis der Auswirkungsanalyse sind die Anforderungen identifiziert, die neu abgesichert werden müssen. Es wird in Abschnitt 3.5 erläutert, wie die zur Absicherung der identifizierten Anforderungen notwendigen Teilsysteme identifiziert und deren **Verhaltensmodelle komponiert** werden. Die Anbindung des technischen Prozesses wird darauffolgend in Abschnitt 3.6 beschrieben.

3.1 Szenario

Eine einfache, diskrete Fertigung ist in Abbildung 2 dargestellt. Ein Werkstück wird auf einem Förderband zu einer Bohrmaschine transportiert, welche ein Loch in das Werkstück bohrt. Das Förderband wird von einem Motor angetrieben. Ein Näherungssensor detektiert das Werkstück, sobald es die Soll-Position unter dem Bohrer erreicht hat. Die verteilten Steuerungskomponenten „Steuerung Bohrer“, „Steuerung Förderband“ und „Hauptsteuerung“ übernehmen Teilaufgaben zur Koordination des Prozessablaufs. „Steuerung Förderband“ steuert das Förderband, „Steuerung Bohrer“ die Bohrmaschine und die „Hauptsteuerung“ koordiniert den Gesamtprozess. Die Aktorik und Sensorik verfügt ebenso über eine Steuerung, über welche die Hardware angesteuert wird. Anhand des Szenarios wird aufgezeigt, wie Änderungen der Steuerungslogik der Komponente „Steuerung Förderband“ zielgerichtet modellbasiert abgesichert werden können.

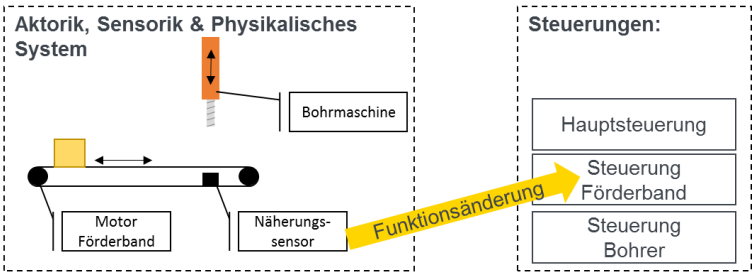

Abbildung 2: Darstellung eines diskreten Fertigungsprozesses

3.2 Komponentenbasierter Modellierungsansatz

Zum automatisierten Aufbau eines Systemmodells wird zunutze gemacht, dass bei Komponenten der Automatisierungstechnik die modellgetriebene Entwicklung zunehmend an Relevanz gewinnt [32]. Dadurch stehen am Ende des Entwicklungsprozesses die Verhaltensmodelle der Komponenten zur Verfügung. Diese Komponentenmodelle sind, verglichen mit dem Systemmodell des gesamten Automatisierungssystems, einfach zu erstellen und zu pflegen. Über Schnittstellen wird das logische Verhalten der Komponente nach außen gekapselt. Die modulare Modellierung ist in Abbildung 3 exemplarisch anhand von zwei Komponenten des Szenarios dargestellt. Die Interaktion der Komponenten mit der Umgebung erfolgt über die Kommunikationsschnittstellen zu anderen Automatisierungskomponenten sowie über Schnittstellen zum technischen Prozess (Aktorik, Sensorik).

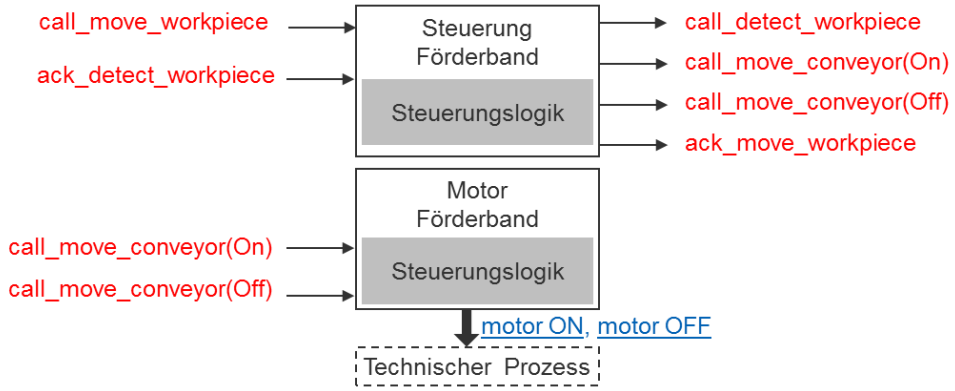

Abbildung 3: Darstellung der Schnittstellen einer Komponente sowie derer gekapselten Steuerungslogik

Es wird angenommen, dass jede Komponente über Anforderungen, die in formaler Sprache vorliegen, verfügt. Diese können sowohl das Soll-Verhalten als auch nichtzulässiges Verhalten bzw.

Systemzustände beschreiben. Zur Verifikation dieser Anforderungen reicht es nicht aus das Verhaltensmodell der Komponente isoliert zu betrachten, sondern es muss in Interaktion mit den Komponenten betrachtet werden, von denen die Komponente abhängig ist. Abhängigkeiten entstehen über Serviceaufrufe (Call-Nachrichten). Durch die Eigenschaft, dass Services gekapselte Funktionseinheiten darstellen, ist ihre Funktionalität nicht abhängig von der aufrufenden Komponente. So entsteht eine logisch hierarchische Anforderungsstruktur zwischen den Komponenten, die sich aus den Aufrufbeziehungen der Komponenten ergibt.

Zur Analyse des Verhaltens eines Teilsystems lässt sich, über die Schnittstellen der Verhaltensmodelle der Komponenten (Komponentenmodelle), ein Verhaltensmodell des Gesamtsystems (Systemmodell) komponieren. Dabei wird sich zunutze gemacht, dass bei Ad-Hoc-Netzwerken die Kommunikationsschnittstellen semantisch beschrieben sind und so die Abhängigkeiten zwischen den Komponenten anhand von deren Schnittstellenbeschreibungen aufgelöst werden können. Auf die Anforderungen an die Verhaltensmodellierung wird im Folgenden eingegangen.

3.3 Verhaltensmodellierung

Wie im Stand der Technik beschrieben, haben sich eine Reihe von Modellierungsarten entwickelt, die zur modellbasierten Verifikation geeignet sind. Das vorgestellte Konzept stellt dabei weitere Anforderungen an die Modellierung:

- Darstellung von parallelen Prozessen
- Schnittstellen zu anderen Automatisierungskomponenten sowie zum technischen Prozess
- Komponierbarkeit
- Abbildbarkeit von zeitlichem Verhalten
- eine formale, mathematisch beschreibbare Darstellung
- eindeutige Abbildbarkeit von Mehrfachzugriff zur Darstellung der Wiederverwendbarkeit von Komponenten durch verschiedene aufrufende Komponenten
- Eingängigkeit und weite Verbreitung

Eine Auswahl an Modellierungsarten ist nach diesen Kriterien in Tabelle 1 bewertet. Soweit eine Modellierungsart den Kriterien in ausreichendem Maß entspricht, kann sie zur Verhaltensmodellierung der Komponenten verwendet werden.

Aufgrund der Eignung für ereignis-gesteuerte Systeme, der einfachen Darstellung von Parallelitäten und der langjährigen Erfahrung am Institut haben sich die Autoren für die Verwendung von offenen Netzen entschieden. Dabei werden offene Netze um Attribute der SIPN erweitert. Die Hinzunahme der Schnittstellenattribute der SIPN schafft die Möglichkeit, die für service-orientierte Architekturen entwickelten offenen Netze zur Modellbildung automatisierter Systeme zu verwenden. Im Folgenden wird beschrieben, wie diese Netze aufgebaut und nach welchen Regeln sie komponiert werden. Darüber hinaus wird betrachtet, wie die Modellierung mit Redundanz und Wiederverwendung von Komponenten umgehen kann.

Definition des offenen Netzes einer Automatisierungskomponente

Das Verhaltensmodell einer Automatisierungskomponente wurde an Definition [33] angelehnt und wird über folgendes 9-Tupel definiert:

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

- $\boldsymbol{P}$: endliche Menge der Stellen des Netzes.
- $\boldsymbol{T}$: endliche Menge der Transitionen des Netzes.

- $\boldsymbol{F}$: endliche Menge der Flussrelationen zwischen Transitionen und Stellen. Es gilt: $F \subseteq (P \times T) \cup (T \times P)$. Die Menge an Vorgängerstellen einer Transition werden über $\cdot t = \{p|(p,t) \in F\}$ beschrieben. Die Nachfolgestellen einer Transition über $t \cdot = \{p|(t,p) \in F\}$. Entsprechendes gilt für die Vorgängertransitionen einer Stelle $\cdot p = \{t|(t,p) \in F\}$ und Nachfolgetransitionen einer Stelle: $p \cdot = \{t|(p,t) \in F\}$.
- $\boldsymbol{M_0}$: Startmarkierung, entspricht einem Vektor der Mächtigkeit |P|.
- $\boldsymbol{\Omega}$: endliche Menge an Markierungen in denen N terminieren darf.
- $\boldsymbol{I}$: Interface-Stellen des Netzes nach außen $I \subseteq P$. Über eine Interface-Stelle werden Nachrichten mit anderen Automatisierungskomponenten ausgetauscht. Die Bezeichnung einer Stelle entspricht dem Namen der Nachricht, die über die Stelle ausgetauscht wird. Dabei gilt für minimale offene Netze:
  - Für Stellen $I1$, über die eine Nachricht versendet wird (Output): $p \cdot = \emptyset$.
  - Für Stellen $I2$, über die eine Nachricht empfangen wird (Input): $\cdot p = \emptyset$.
  - Eine Interface-Stelle I darf nicht gleichzeitig Input und Output sein: $I1 \cap I2 = \emptyset$.
- **A**: Schnittstelle zum technischen Prozess. Dabei werden Aktionen auf das technische System beim Belegen einer Stelle ausgeführt. Es gilt: $|A| \equiv |P|$.
- **S**: Schnittstelle vom technischen Prozess. Dabei werden durch Aktionen vom technischen Prozess Transitionen geschaltet. Es gilt: $|S| \equiv |T|$.
- $\boldsymbol{L}$: Latenzzeit der Kanten. Es gilt: $|L| \equiv |F|$.

Die graphische Darstellung eines offenen Netzes ist exemplarisch an der Komponente „Steuerung Förderband“ $N_{Förderb.}$ in Abbildung 4 aufgezeigt. Dies ist dem Szenario aus Abbildung 2 entlehnt. Die Steuerung des Förderbands koordiniert den Motor des Förderbands sowie den Näherungssensor. Die Interface-Stellen sind über rote Kreise dargestellt und befinden sich auf der gestrichelten Systemgrenze der Komponente. Bei den Stellen auf der linken Systemgrenze handelt es sich um Input-Stellen und bei den Stellen auf der rechten Systemgrenze um Output-Stellen. Der Ablauf der Steuerung ist dabei wie folgt: Initial befindet sich die Komponente im Zustand „FördernAUS“. Bei Aufruf des Services der Komponente („callFörderband(On)“) wird ein Token in der Stelle „callFörderband(On)“ erzeugt. Die Transition T1 wird schaltfähig. Wie bei SIPN üblich, schaltet eine Transition, sobald sie schaltfähig ist. Die Tokens der belegten Stellen „FördernAUS“ und „callFörderband(On)“ werden abgezogen und Tokens in der Stelle „FördernAN“ sowie in den Output-Stellen „callNäherung“ und „callFMotor(On)“ erzeugt. Das Belegen der Output-Stellen durch ein Token bewirkt das Senden eines Service-Aufrufs. In diesem Fall werden die Services des Näherungssensors sowie des Förderbandmotors aufgerufen. Bestätigt der Näherungssensor, dass das Werkstück detektiert wurde, wird ein Token in der Stelle „ackNäherung“ erzeugt, die Transition T2 wird schaltfähig und die Steuerung des Förderbands geht wieder in den Zustand „FördernAUS“. Dabei wird der Förderbandmotor deaktiviert „callFMotor(Off)“ und das Ende des Steuerungsprogramms wird gemeldet „ackFörderband“.

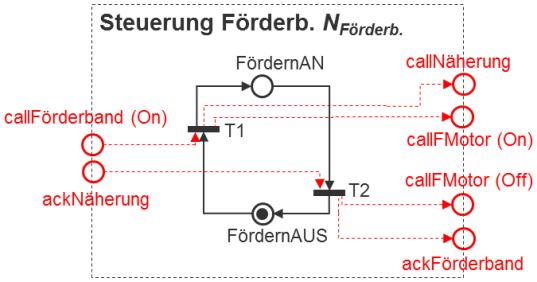

Abbildung 4: Hauptsteuerung einer diskreten Fertigung beschrieben über ein offenes Petri-Netz

Neben Kommunikationsschnittstellen können Komponenten über Schnittstellen zum technischen Prozess verfügen. Die Schnittstellen sind graphisch durch eine unterstrichene Bezeichnung gekennzeichnet. Dabei werden, analog zur Beschreibung von SIPN, von Stellen Aktionen auf den technischen Prozess ausgeführt. Das Automatisierungssystem wird über Transitionen durch Ereignisse des technischen Prozesses beeinflusst. In Abbildung 5 links ist dies anhand des Förderbandmotors und in Abbildung 5 rechts anhand des Näherungssensors aufgezeigt.

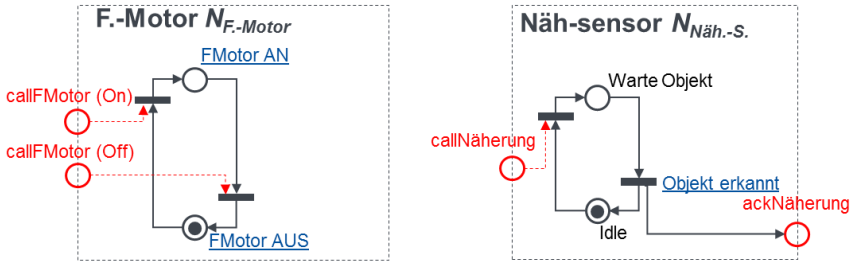

Abbildung 5: Schnittstelle zum technischen Prozess (links), Schnittstelle vom technischen Prozess (rechts)

Liegt von jeder Komponente des Automatisierungssystems das Komponentenmodell in Form eines offenen Netzes vor, kann ein Systemmodell komponiert werden. Dies ist notwendig, um solche Anforderungen modellbasiert verifizieren zu können, die sich auf mehrere Komponenten beziehen.

Zur Erstellung des Systemmodells werden die offenen Netze der Komponente über die Interface-Stellen zu einem zusammenhängenden Netz komponiert. Die Komposition N offener Netze ($N_{Komp.1} \dots N_{Komp.n}$) erfolgt über folgende Rechenregeln.

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

Die Komposition der Verhaltensmodelle der Komponenten „Steuerung Förderband“, „Motor Förderband“ und „Näherungssensor“ ist in Abbildung 6 graphisch dargestellt.

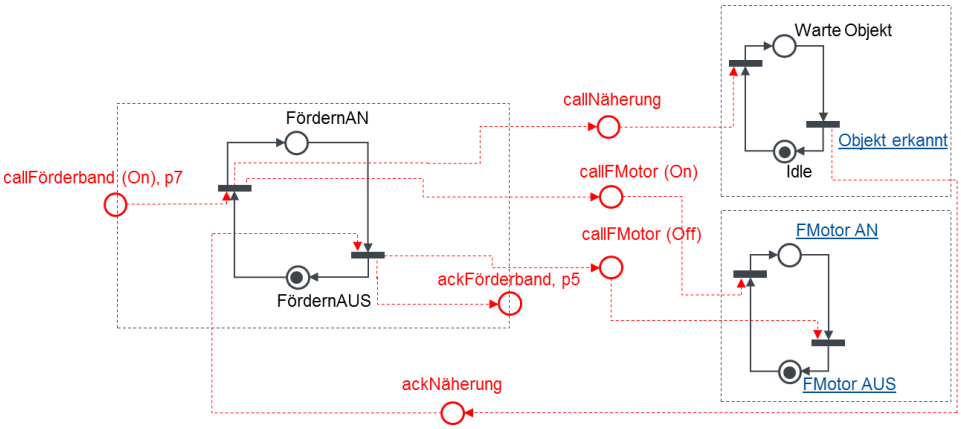

Abbildung 6: Komposition der Verhaltensmodelle dreier Komponenten. Die Steuerung des Förderbands ist links, der Näherungssensor rechts oben und der Förderbandmotor rechts unten dargestellt.

Über die Interface-Stellen lassen sich bei Redundanz im System mithilfe der beschriebenen Rechenoperationen weitere offene Netze ankoppeln. Weitere Komponenten, welche den gleichen Service anbieten, stellen die Redundanz des Systems dar. Nach dem Prinzip der Wiederverwendbarkeit können Komponenten von verschiedenen Komponenten aufgerufen werden. In diesem Fall ist die Eindeutigkeit des Gesamtsystems nicht mehr gegeben. Abbildung 7 verdeutlicht dies, indem angenommen wird, dass neben der Steuerung des Förderbandes eine zweite Komponente (Steuerung B) auf den Näherungssensor zugreift. Ruft „Steuerung Förderband“ den Service des Näherungssensors auf, wandert ein Token über die „call“-Stelle in das offene Netz des Näherungssensors. Nach Detektion eines Werkstücks wird der Token wieder über die „ack“-Stelle an die aufrufende Komponente zurückgegeben. Dabei ist bei der Verzweigung der „ack“-Stelle nicht mehr unterscheidbar, welche Komponente die aufrufende war. Befindet sich Steuerung B ebenfalls in einem Zustand, indem Sie schaltbereit ist, könnte der Token fälschlicherweise von Steuerung B abgezogen werden. Um dies zu verhindern, wird das Netz bei der Komposition der Netze um farbige Token erweitert, falls mehrere Komponenten die gleiche Komponente aufrufen. Wie in Abbildung 7 dargestellt, wird das Token bei Aufruf des Näherungssensors durch „Steuerung Förderband“ um das Attribut ID erweitert, welches den Wert A zugewiesen bekommt. Dieses farbige Token wird weitergegeben. An der „ack“-Stelle wird anhand des Attributs entschieden, zu welcher Komponente das Token weiter gereicht wird. Das Attribut wird nach Durchführung des Services von der aufrufenden Komponente wieder entfernt. Diese eindeutige ID der Komponenten wird automatisiert in das komponierte Netz integriert und bedarf keiner Anpassung an der Modellierung der Komponenten.

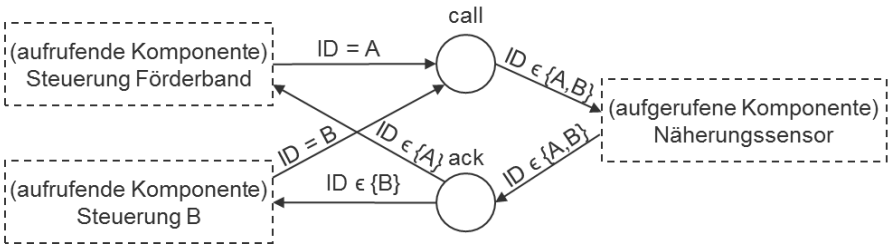

Abbildung 7: Erweiterung des offenen Netzes um farbige Token bei mehreren aufrufenden Komponenten einer Komponente

Die Modellierung ist die Grundlage für das dreistufige Konzept aus Abbildung 1, welches im Folgenden erläutert wird.

3.4 Auswirkungsanalyse

Zur Eingrenzung der Auswirkungen von Funktionsänderungen werden Informationen über die Abhängigkeiten zwischen Komponenten des Automatisierungssystems benötigt. Zur Identifikation der von Softwareänderungen betroffenen Anforderungen werden aus den Komponentenmodellen Abhängigkeitsinformationen extrahiert und ein Strukturdiagramm zur graphischen Darstellung aufgebaut. Das Blockdefinitionsdiagramm, welches an das Klassendiagramm der UML angelehnt ist, eignet sich aufgrund der Standardisierung in SysML und der weiten Verbreitung sehr gut. Die Abhängigkeiten zwischen den Komponenten entstehen, wenn zur Ausführung der Funktionalität einer Komponente auf die Funktionalität einer anderen Komponente zugegriffen wird. Dies ist der Fall, wenn eine Komponente den Service einer weiteren aufruft (Call-Service). Gemäß der vorab beschriebenen logisch hierarchischen Anforderungsdefinition ist eine serviceorientierte Komponente nicht von der Komponente abhängig, die sie aufruft. Das Blockdefinitionsdiagramm des Szenarios ist in Abbildung 8 dargestellt. Dabei zeigt der gestrichelte Pfeil von der abhängigen Komponente auf die unabhängige Komponente.

Die Abhängigkeitsinformationen zum Aufbau des Blockdefinitionsdiagramms werden aus den Komponentenmodellen extrahiert. Dabei wird anhand der Bezeichnungen der Interface-Stellen geprüft, ob sie einem Service-Aufruf (call) entsprechen. Handelt es sich dabei um einen Output stellt die jeweilige Komponente den Ursprung eines Abhängigkeitspfeils dar. Entspricht die Interface-Stelle dem Input eines Service-Aufruf (call), mündet der Abhängigkeitspfeil in die jeweilige Komponente.

Findet bei einer Komponente eine Funktionsänderung statt, ist nicht mehr sichergestellt, dass die Anforderungen der abhängigen Komponenten noch erfüllt sind. Dies resultiert darin, dass die Anforderungen der Komponenten betroffen sind, die entgegen der Pfeilrichtung der Abhängigkeitspfeile von den Komponenten erreichbar sind.

Anhand von Abbildung 8 wird verdeutlicht, dass bei einer Funktionsänderung der Komponente „Steuerung Förderband“ nach dem beschriebenen Vorgehen die Anforderungen der Komponente „Steuerung Förderband“ und „Hauptsteuerung“ erneut verifiziert werden müssen.

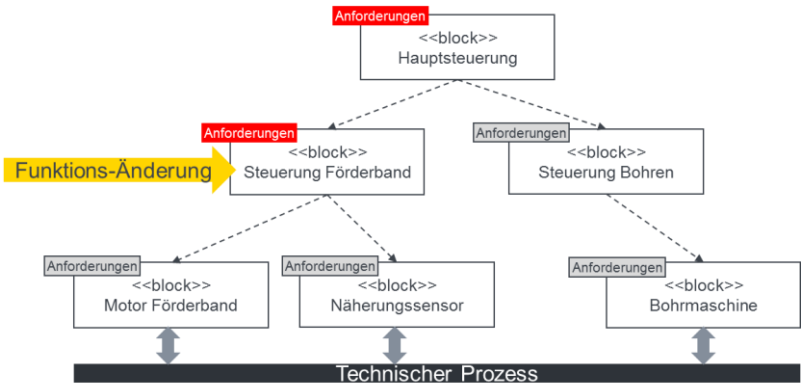

Abbildung 8: Blockdefinitionsdiagramm des automatisierten Systems

3.5 Komposition der zur Verifikation notwendigen Teilsysteme

Sind die zu verifizierenden Anforderungen erkannt, wird in einem zweiten Schritt das jeweils zur modellbasierten Verifikation notwendige Verhaltensmodell aufgebaut.

Zur formalen Verifikation der Anforderungen, die einer Komponente zugeordnet sind, wird ein Modell benötigt, das die Funktionalität der Komponente abbildet. Wie vorab beschrieben muss zur Absicherung der Anforderungen einer Komponente die Interaktion mit den Komponenten berücksichtigt werden, von denen sie abhängig ist. Das entspricht den Komponenten, die in Pfeilrichtung erreichbar sind. So entsteht für die Anforderungen jeder Komponente ein individuelles, „maßgeschneidertes“ Verhaltensmodell, das auf Basis der vorgestellten Formeln komponiert werden kann.

Zur Verifikation der Anforderungen der Komponente „Steuerung Förderband“ sind somit die Modelle der Komponenten notwendig, von denen „Steuerung Förderband“ abhängig ist. Durch Folgen der Abhängigkeitspfeile des Blockdefinitionsdiagramms in Pfeilrichtung lassen sich die Komponenten „Motor Förderband“ und „Näherungssensor“ ermitteln (siehe Abbildung 9 links). Die Komposition der offenen Modelle ist in Abbildung 9 rechts dargestellt.

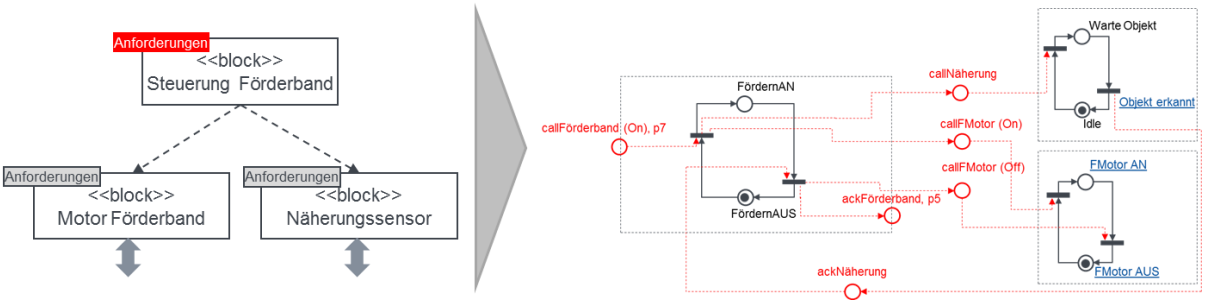

Abbildung 9: Zur Simulation der Anforderungen der Komponente "Steuerung Förderband" notwendige Komponenten (links), Teilsystem zur Beschreibung des Verhaltens der „Steuerung Förderband“ notwendigen Komponenten (rechts)

Das reduzierte Teilsystem lässt sich nun gegen die Anforderungen der betroffenen Komponenten effizient modellbasiert verifizieren. Dazu benötigt es noch Treiber, welche die zu prüfende Funktionalität aufrufen. Die Treiber können durch Erweiterung der Startmarkierung automatisiert generiert werden. Dabei wird ein zusätzliches Token in der Interface-Stelle erzeugt, welche die zu prüfende Funktionalität ausführt. Weitere ungebundene Stellen bleiben erhalten. Ungebundene Input-Stellen beschreiben Services, die bei der Ausführung der zu prüfenden Funktionalität nicht aufgerufen werden. Dies ermöglicht einen sauberen Abschluss des komponierten Verhaltensmodells zu benachbarten Verhaltensmodellen. Dieser Umstand bietet, neben den formalisierten Anforderungen und dem Modell des technischen Prozess, die Eingangsinformationen für modellbasierte Verifikationsverfahren.

3.6 Anbindung des technischen Prozesses

Der Fokus des Artikels liegt auf der Absicherung von Software-Änderungen, weshalb bisher hauptsächlich die Abhängigkeitsbeziehung über Kommunikationsschnittstellen betrachtet wurde. Da Komponenten zusätzlich über den technischen Prozess interagieren, muss dieser ebenso als Verhaltensmodell abgebildet werden. Wie bei Integrationstests üblich, reicht es aus, die Komponenten, die nicht im Fokus der Betrachtung liegen, als Platzhalter zu realisieren [9]. Die Abbildung der logischen Abhängigkeiten zwischen Aktorik und Sensorik sowie der aktuelle Zustand des

technischen Prozesses sind die wesentlichen Eigenschaften, die ein Platzhalter erfüllen muss. Prozessinterpretierbare Petri-Netze (PIPN) können dazu verwendet werden. Ein möglicher Aufbau des Platzhalters ist in Abbildung 10 dargestellt. Die Anbindung der Steuerung und des technischen Prozesses erfolgt dabei über Testkanten, die analog zu den Schnittstellen innerhalb des technischen Prozesses automatisiert komponiert werden können. Testkanten unterscheiden sich von üblichen Kanten dahingehend, dass von der Stelle beim Schalten der Transition kein Token abgezogen wird.

Der benötigte Platzhalter ist abhängig von den Anforderungen, die es zu verifizieren gilt. Bei modularen Produktionssystemen kann aus Modulen des technischen Prozesses der notwendige Teilprozess komponiert werden. Eine Topologie erlaubt dabei die Realisierung der Schnittstellen zwischen den Teilprozessen. Aktuell werden die notwendigen Teilprozesse sowie deren Anfangsmarkierungen mit den Anforderungen der Komponente definiert. Dies wurde prototypisch für den nachfolgend vorgestellten Demonstrator entworfen. In diesem Bereich sind weitere Untersuchungen für allgemeingültige Konzepte notwendig.

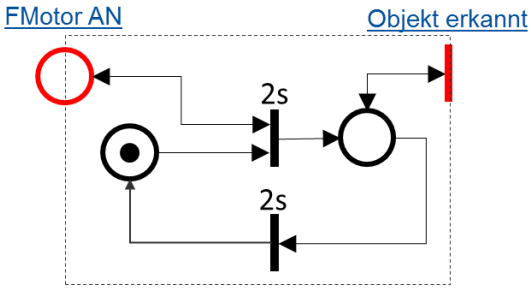

Abbildung 10: Technischer Prozess modelliert als PIPN

4 Evaluierung anhand von „TestIAS“

Die Evaluierung des beschriebenen Konzeptes wird anhand von „TestIAS“ realisiert, welches in Abbildung 11 dargestellt ist. TestIAS verfügt über Schnittstellen zum Automatisierungssystem und zu Datenbanken in welchen die Komponentenmodelle sowie die Systemanforderungen verwaltet werden (Modell-Pool). Eine OPC UA Schnittstelle ermöglicht die Ad-hoc Integration in das Automatisierungssystem.

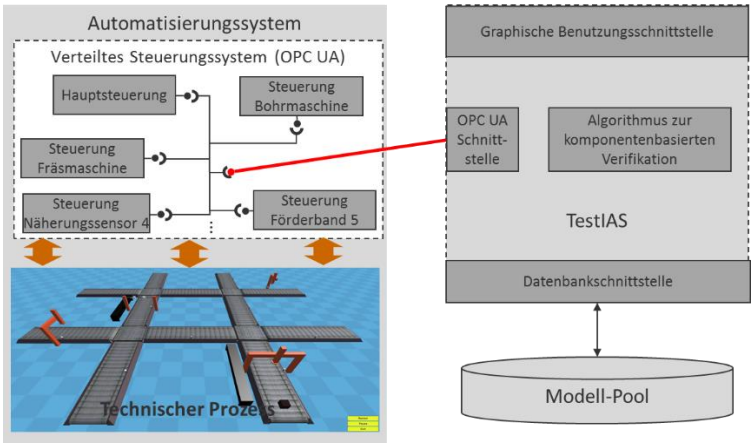

Abbildung 11: TestIAS

Im Rahmen dieses Szenarios werden die Modelle des Modell-Pools von den Herstellern der Komponenten und den Anlagenbauern zur Verfügung gestellt.

Das Automatisierungssystem besteht aus einem OPC-UA basierten, dezentralen Steuerungssystem sowie einem dadurch gesteuerten technischen Prozess. Das Automatisierungssystem ist wesentlich komplexer als in dem vorgestellten Szenario. Das Steuerungssystem besteht aus 153 Services, die auf sechs OPC-UA-Server verteilt sind und den Produktionsprozess koordinieren. Zwischen den Komponenten bestehen insgesamt 171 Abhängigkeiten. In Abbildung 11 sind beispielhaft fünf Services dargestellt. Eine Konfigurationsschnittstelle erlaubt die Änderung der Funktionalität der Services. Der durchschnittliche Vernetzungsgrad der Services beträgt nach der Fan-In/Fan-Out Metrik 4,06. Die zyklomatische Komplexität der einzelnen Komponenten nach McCabe erreicht Werte bis 35. Somit besitzt das entstehende Modell eine Größe, welche durch manuelle Tätigkeiten von Menschen kaum beherrschbar ist. Der technische Prozess, welcher eine diskrete Fertigung darstellt, wird durch eine Simulation umgesetzt. Die Simulation basiert auf einer Game Engine und interagiert über in Netzwerknachrichten eingebettete Aktor- und Sensorsignale mit der Steuerung. Die Integration einer Virtual Reality Brille stellt die Simulation realer dar. Für das modulare Produktionssystem wurde eine Topologie entworfen über die die Modelle der Teilprozesse automatisiert und Interface-Stellen komponiert werden.

TestIAS registriert sich ad-hoc bei Anschluss an das Automatisierungssystems als OPC UA Client. Durch Auslesen des Serviceverzeichnisses des Automatisierungssystems (Discovery-Server) identifiziert TestIAS die im Automatisierungssystem angemeldeten Services. Jeder Service verfügt über eine eindeutige ID sowie über ein Versionierung. Identifiziert TestIAS Änderungen des Automatisierungssystems, startet der im Konzept beschriebene Ablauf. Die notwendigen Informationen werden aus dem Modell-Pool bezogen. Die Petri-Netze liegen im XML-basierten Austauschformat .PNML vor, die Anforderungen im .CTL-Format. Ist die Verifikation beendet, können die Ergebnisse und Zwischenergebnisse eingesehen werden. Auf der in Abbildung 12 dargestellten graphischen Benutzungsschnittstelle können die Verifikationsergebnisse der geprüften Anforderungen eingesehen werden. So lässt sich durch Abbildung 12 nachvollziehen, welche Anforderungen nicht eingehalten wurden. Anhand des darunter dargestellten komponierten Verhaltensmodells können Fehler in den Modellen identifiziert werden. In der linken Spalte kann der Nutzer zusätzlich die graphische Darstellung der Einzelmodelle sowie das komponierte Gesamtmodell einsehen. Mithilfe eines automatisiert aufgebauten Blockdefinitionsdiagramms, werden die Abhängigkeiten im System nachvollziehbar. Die korrekte Verifikation von gültigen sowie ungültigen Konfigurationen konnte an definierten Software-Änderungen bereits gezeigt werden. Die Änderungsszenarien berücksichtigen die Änderung eines Produkttyps, die Änderung eines Steuerungsprozesses einer Produktionsmaschine sowie Änderungen an dem Verhalten von Aktorik und Sensorik. Die daraus resultierenden Fehlerwirkungen wurden durch Deadlocks in der Steuerung oder dem Nichterreichen von definierten Zuständen sichtbar. Dabei traten Fehlerwirkungen entweder in der geänderten Komponente auf oder in Komponenten die von ihr Abhängig sind. Das größte komponierte Petri-Netz besteht nach der Entfaltung aus 476 Stellen. Die Verifikation des Petri-Netzes benötigt mit dem Model-Checker „ITS-Tool“ auf einem i7-Prozessor mit einer Taktfrequenz von 3,6 GHz unter Nutzung eines Rechenkerns 64 Minuten. Bei großen Systemen kann der zur Verifikation notwendige Rechenaufwand einen limitierenden Faktor darstellen. Durch die Kapselung von Automatisierungssystemen ist aber bei dezentralen Systemen nicht zu erwarten, dass die Komplexität der Systeme ins Unendliche wächst. Es wird dadurch auch ersichtlich, wie elementar eine Auswirkungsanalyse ist, um die betroffenen Teilsysteme einzugrenzen. Die Verifikation weiterer Petri-Netze kann parallel auf anderen Prozessorkernen berechnet werden.

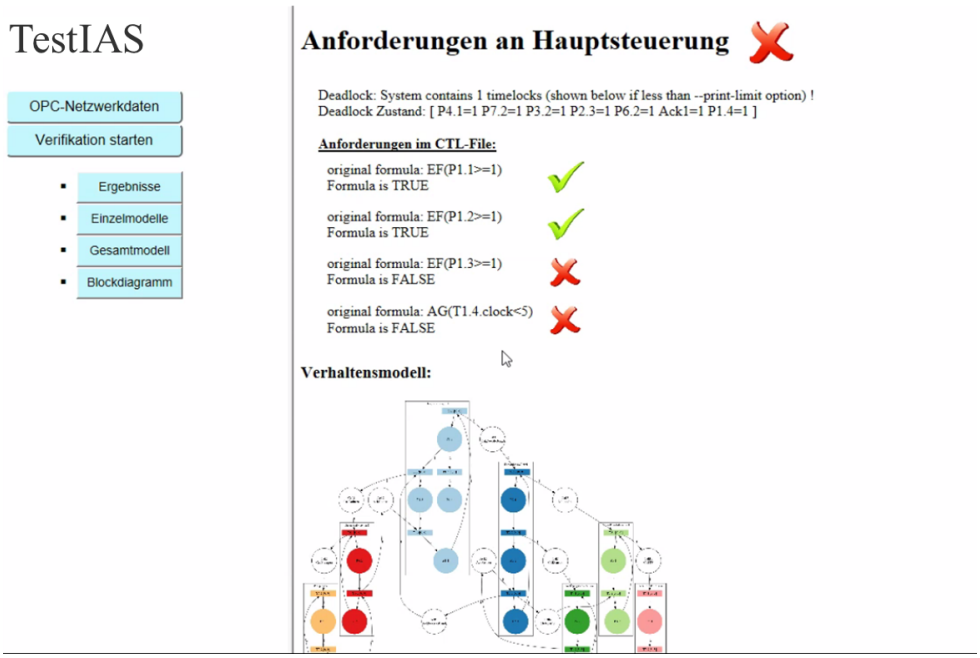

Abbildung 12: Graphische Benutzungsschnittstelle von TestIAS

5 Zusammenfassung und Ausblick

Zur zielgerichteten Absicherung von Änderungen ist die Kenntnis über die Abhängigkeiten zwischen den Komponenten des Automatisierungssystems unerlässlich. Da sich die Änderungen auch auf die Abhängigkeiten zwischen den Komponenten auswirken können, stehen aktuelle Systemmodelle, welche diese Abhängigkeiten beinhalten, in der Regel nicht zu Verfügung.

Anhand eines Konzepts wurde aufgezeigt, wie ohne vorliegendes Systemmodell Änderungen an verteilten Automatisierungssystemen durch Anlagenbetreiber zielgerichtet abgesichert werden können. Als Basis dienen Verhaltensmodelle der einzelnen Komponenten des Automatisierungssystems, welche analysiert und komponiert werden. Daraus lassen sich die von Änderungen betroffenen Anforderungen ermitteln und ein Verhaltensmodell des Teilsystems erstellen, welches zur modellbasierten Verifikation der Anforderungen benötigt wird. Die Eingrenzung betroffener Anforderungen und die Komposition des zur Verifikation notwendigen Teilmodells bilden die Grundlage für eine effiziente Verifikation.

Realisiert und evaluiert wird das Konzept mit Hilfe von TestIAS. TestIAS lässt sich ad-hoc in ein verteiltes Automatisierungssystem (OPC-UA basiert) integrieren, analysiert automatisiert die Auswirkungen von Änderungen und verifiziert diese modellbasiert. Dies erlaubt die modellbasierte Verifikation von verteilten Automatisierungssystemen auf einer logischen Ebene. Anhand verschiedener Änderungsszenarien konnte die Anwendbarkeit des Verfahrens gezeigt werden. Das verteilte Automatisierungssystem besitzt eine Komplexität, welche durch rein manuelle Tätigkeiten schwer beherrschbar ist. Die Verifikationsergebnisse sowie entstandene Modelle werden für den Benutzer anschaulich dargestellt.

In weiteren Untersuchungen soll aufgezeigt, dass das Konzept auf andere Automatisierungssysteme übertragbar ist. Des Weiteren soll die einfache Erweiterung für den Fall gezeigt werden, dass das definierte 9-Tupel zur Modellierung der Komponenten nicht ausreicht. Dabei soll verdeutlicht werden, dass sich dieser Umstand lediglich auf die Komplexität des Modells auswirkt.

6 Literatur

[1] Vogel-Heuser, Birgit; Fay, Alexander: Evolution of software in automated production systems: Challenges and research directions, The Journal of Systems and Software 110 54-84, 2015.

[2] Forschungsunion; Acatech: Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0, 2013.

[3] Siepmann, David: Industrie 4.0 – Fünf zentrale Paradigmen in: Einführung und Umsetzung von Industrie 4.0, hg von Armin Roth, Springer-Verlag 2016.

[4] Fay, Alexander, Vogel-Heuser, Birgit; Frank, Timo; Eckert, Karin; Hadlich, Thomas; Diedrich, Christian: Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns, The Journal of Systems and Software 101 (2015) 221-235.

[5] Göhner, Peter: Softwareagenten für die flexible Kopplung von Automatisierungssystemen. 6. VDI-Expertenforum "Agenten im Umfeld von Industrie 4.0", München, 2014.

[6] Zeller, Andreas; Weyrich, Michael: Challenges for Functional Testing of reconfigurable Production Systems, 21st IEEE International Conference on Emerging Technologies and Factory Automation, Berlin, 2016.

[7] Zeller, Andreas; Weyrich, Michael: Industrie 4.0 mit vernetzter und flexibler Produktion erfordert neue Testmethodiken, atp edition, S. 16-18, 10/2015.

[8] Legat, Christoph; Steden, Frank; Feldmann, Stefan; Weyrich, Michael; Vogel-Heuser, Birgit: Co-Evolution and Reuse of Automation Control and Simulation Software, IECON 2014-40th Annual Conference of the IEEE, 2014.

[9] ISTQB – International Software Testing Board: Certified Tester Foundation Level Syllabus, Version 2011 .10.1, hg von: Austrian Testing Board, German Testing Board e.V. & Swiss Testing Board, 2011.

[10] Krause, Jan: Testfallgenerierung aus modellbasierten Systemspezifikationen auf Basis von Petrinetzentfaltungen. Shaker Verlag Aachen, 2012.

[11] Khlifi, Oussama; Mosbahi, Olfa; Khalgui, Mohamed; Frey Georg: New Verification Approach for Reconfigurable Distributed Systems: ICSOFT-2017-12th International Conference on Software Technologies, Madrid, 2017.

[12] Schlich, Bastian, Brauer, Jörg, Wernerus, Jörg, Kowalewski, Stefan: Direct Model Checking of PLC Programs in IL, 2nd IFAC Workshop on Dependable Control of Discrete Systems DCDS'09

[13] Blech, Jan Olaf; Lindgren, Per; Pereira, David; Vyatkin, Valeriy; Zoitl, Alois: A Comparison of Formal Verification Approaches for IEC 61499; IEEE International Conference on Emerging Technologies and Factory Automation, Berlin, Germany, 2016.

[14] Broy, Manfred; Fox, Jorge; Hölzl, Florian; Koss, Dagmar; Kuhrmann, Marco; Meisinger, Michael; Penzenstadler, Birgit; Rittmann, Sabine; Schätz, Bernhard; Spichkova, Maria; Wild, Doris: Service-oriented Modeling of CoCoME with Focus and AutoFocus, The Common Component Modeling Example, Springer, 2008.

[15] Spichkova, Maria: Focus on Isabelle: From specification to verification, Technical Report Department of Electrical and Computer Engineering, Concordia University, 2008.

[16] Legat, Christoph; Mund, Jakob; Campetelli Alarico; Hackenberg, Georg; Folmer, Jens; Schütz, Daniel; Broy, Manfred; Vogel-Heuser, Birgit: Interface Behavior Modeling for Automatic Verification of Industrial Automation Systems' Functional Conformance, Automatisierungstechnik (at), 62(11):815—825, 2015.

[17] Ladiges, Jan; Haubeck, Christopher; Fay, Alexander; Lamersdorf, Winfried: Evolution Management of Production Facilities by Semi-Automated Requirement Verification, Automatisierungstechnik (at), 62(11):781—793, 2015.

[18] Lochau, Malte; Mennicke, Stephan; Baller, Hauke; Ribbeck Lars: Incremental model checking of delta-oriented software product lines, Journal of Logical and Algebraic Methods in Programming 85 245–267, 2016.

[19] https://www.3ds.com/products-services/catia/products/dymola, abgerufen am:2017.03.03.

[20] Behrmann Gerd, David Alexandre, Larsen Kim: A Tutorial on UPPAAL, in: Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Science, hg von Bernardo Marco, Springer-Verlag 2004.
[21] Vogel-Heuser, Birgit; Folmer, Jens; Frey, Georg; Liu, Liu; Hermanns, Holger; Hartmanns, Arnd: Modeling of Networked Automation Systems for Simulation and Model Checking of Time Behavior, 9th International Multi- Conference on Systems Signals and Devices, Chemnitz, Germany, 2012.
[22] ISO/IEC 15909-1: 2004-12, system and software engineering - High-level Petri nets - Part 1: Concepts, definitions and graphical notation.
[23] ISO/IEC 15909-2:2011-02, Systems and software engineering -- High-level Petri nets -- Part 2: Transfer format.
[24] Rausch, Mathias; Hanisch, Hans-Michael: Netz Condition/Event System with Multiple Condition Outputs, p. 592–600, in: Symposium on Emerging Technologies and Factory Automation, vol. 1, 1995.
[25] Khalgui, Mohamed: NCES-based modelling and CTL-based verification of reconfigurable embedded control systems, Computers in Industry 61, p. 198-212, 2010.
[26] Hanisch, Hans-Michael; Vyatkin, Valeriy: Verification of distributed control systems in intelligent manufacturing, Journal of Intelligent Manufacturing, 14, p. 123-136, 2003.
[27] Aalst, Will; Lohmann, Niels; Massuthe, Peter; Stahl, Christian; Wolf, Karsten: Multiparty Contracts: Agreeing and Implementing Interorganizational Processes, The Computer Journal, 2010.
[28] Frey, Georg: Hierarchical design of logic controllers using signal interpreted Petri nets, IFAC Proceedings Volumes 36 Issue 6, p. 361-366, 2003.
[29] IEC 61131-3:2013-03, Speicherprogrammierbare Steuerungen – Teil 3: Programmiersprachen.
[30] Biallas, Sebastian: Verification of Programmable Logic Code using Model Checking and Static Analysis, Dissertation RWTH Aachen Department of Computer Science, Technical Report, 2016.
[31] Rösch, Susanne; Ulewicz, Sebastian; Provost, Julien; Vogel-Heuser, Birgit: Review of Model-Based Testing Approaches in Production Automation and Adjacent Domains - Current Challenges and Research Gaps, Journal of Software Engineering and Applications, p. 499-519, 08/2015.
[32] Vogel-Heuser, Birgit; Schütz, Daniel; Frank, Timo; Legat, Christoph: Model-driven Engineering of Manufacturing Automation Software Projects – A SysML-based approach, Mechatronics, 2014.
[33] Kindler, Ekkart: „A Compositional Partial Order Semantics for Petri Net Components“, ICATPN, p. 235–252, 1997.