Methoden

Jan Ladiges, Aljosha Köcher, Peer Clement, Henry Bloch, Thomas Holm, Paul Altmann, Alexander Fay* und Leon Urbas

Entwurf, Modellierung und Verifikation von Serviceabhängigkeiten in Prozessmodulen

Design, modelling and verification of service dependencies in process modules

https://doi.org/10.1515/auto-2017-0076
Empfangen 19. Juni 2017; angenommen 7. Februar 2018

Zusammenfassung: Dieser Beitrag beschreibt eine Methode für das Engineering von Abhängigkeiten zwischen Services, die ein Modul einer Produktionsanlage in der Prozessindustrie anbietet. Hierzu wird eine Serviceabhängigkeitsmatrix eingeführt, die beim Engineering eines Moduls genutzt werden kann, um die Abhängigkeiten zwischen Services festzulegen. Die Services werden jeweils als Petri-Netz beschrieben und durch die Abhängigkeiten in eine zusammenhängende Petri-Netz-Struktur überführt. Zur effizienten Darstellung der Services und ihrer Abhängigkeiten wird, basierend auf dem PNML-Kernmodell, ein neuer Petri-Netz-Typ eingeführt, die ServiceNets. Eine formale Analyse dieser Petri-Netze erlaubt die Verifikation u. a. hinsichtlich erreichbarer Zustände und Deadlock-Freiheit.

Schlagwörter: Modulare Prozessautomation, Serviceabhängigkeiten, Modulengineering, Module Type Package, Petri Net Markup Language

Abstract: The capabilities of modules in modular plants in the process industry can be described by services. To describe the dependencies between such services, a service dependency matrix is introduced in this contribution. Each service is modelled by a seperate Petri Net first, which are then linked by the dependencies. For an efficient modelling of this larger Petri Net, so-called Service Nets are introduced on the basis of a PNML core model. By means of an analysis, the system can be verified regarding the presence/absence of deadlocks and other undesired characteristics.

Keywords: modular process automation, service dependencies, module engineering, Module Type Package, Petri Net Markup Language

***Korrespondenzautor: Alexander Fay,** Helmut-Schmidt-Universität, Hamburg, Germany, E-Mail: alexander.fay@hsu-hh.de
Jan Ladiges, Aljosha Köcher, Peer Clement, Henry Bloch, Helmut-Schmidt-Universität, Hamburg, Germany, E-Mails: jan.ladiges@hsu-hh.de, aljosha.koecher@hsu-hh.de, peer.clement@hsu-hh.de, henry.bloch@hsu-hh.de
Thomas Holm, WAGO Kontakttechnik GmbH & Co. KG, Minden, Germany, E-Mail: Thomas.Holm@wago.com
Paul Altmann, Leon Urbas, Technische Universität Dresden, Dresden, Germany, E-Mails: paul.altmann@tu-dresden.de, leon.urbas@tu-dresden.de

1 Einleitung

1.1 Motivation und Grundlagen der modularen Prozessautomation

In der Prozessindustrie erhöhen volatile Beschaffungs- und Absatzmärkte sowie der zunehmende Wunsch nach kundenindividuellen Produkten den Bedarf an wandlungsfähigen Produktionsanlagen [1]. Gleichzeitig wird angestrebt, die Zeitspanne von der Idee neuer oder angepasster Produkte bis zur Markteinführung signifikant zu verkürzen [2]. Entsprechend werden Bemühungen unternommen, die zu einer Flexibilisierung des Produktionsprozesses sowie zu einer Reduzierung der *Time-to-Market* führen sollen.

Ein insbesondere in der Chemie-, Pharma- und Nahrungsmittelindustrie vielversprechender Ansatz, Prozessanlagen mit der nötigen Wandlungsfähigkeit auszustatten, ist die *Modularisierung* [3]. Modulare Prozessanlagen setzen sich aus Modulen zusammen, die jeweils definierte (Grund-)Funktionen ausführen können. Diese Funktionen, wurden vorab in der Modulautomatisierung implementiert und können über eine einheitliche Schnittstelle von einem übergeordneten System, der *Prozessführungsebene* (*PFE*), aufgerufen, parametriert und koordiniert werden [3]. Der Vorteil gegenüber monolithisch errichteten Anlagen ist ein Gewinn an Flexibilität hinsichtlich des Produktionsprozesses mittels der Möglichkeit der kurzfristigen Ergänzung oder des Austauschs von Modulen sowie einer aufwandsarmen Anpassung der Modul-

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

konfiguration und -Parametrierung. Zudem ist es möglich mittels *numbering-up* der Modulanzahl (als Gegenstück zum heute üblichen *scale-up*) eine dynamische Anpassung der Produktionskapazität vorzunehmen [4].

Ein weiterer Vorteil ergibt sich daraus, dass die Module nicht erst im Zuge des Engineering der zu errichtenden Produktionsanlage entworfen, errichtet und getestet werden, sondern bereits in einem zeitlich vorgelagerten Modulengineering-Prozess. Das Engineering der Produktionsanlage reduziert sich somit auf die Integration der Module zur Gesamtanlage und zur Projektierung der Inter-Modul-Abhängigkeiten und -Funktionalitäten. Ein Großteil der anfallenden Engineeringleistung verfahrenstechnischer Anlagen wird somit bereits im Modulengineering erbracht, was eine signifikante Reduktion der projektabhängigen Engineering-Aufwände für die Anlage zur Folge hat [4]. Es ergibt sich somit eine Zweiteilung des ursprünglichen Engineeringprozesses in ein vorgelagertes und projektunabhängiges Modulengineering sowie in ein projektabhängiges Integrationsengineering [4]. Sind Module zudem universell entworfen, um für verschiedene Produktionsprozesse in unterschiedlichen Anlagen einsetzbar zu sein, ist eine Wiederverwendung der Module möglich [5], was eine nochmalige Verringerung der Gesamtaufwände zur Folge hat. Neben der o. g. Flexibilität, die durch die Möglichkeit der aufwandsarmen Rekonfigurierbarkeit mittels Anpassung der Modulkonfiguration hervorgerufen wird, verspricht die Modularisierung also vor Allem eine verkürzte Time-to-Market neuer oder geänderter Produkte [6].

Der Grad an Flexibilität bzw. die Verkürzung der Dauer zur Errichtung und Anpassung einer Anlage ist somit stark abhängig von dem Aufwand, der für das Integrationsengineering unternommen werden muss. Generell wird hier eine schnelle und unkomplizierte Integration der Module angestrebt. Aus der Betrachtung der Automatisierung modularer Anlagen ergibt sich demnach die Notwendigkeit einer Methode zur logischen Integration und Abbildung der Modulfunktionalitäten, der Modulbedienung und der dafür notwendigen Kommunikation, in die PFE. Eine weitestgehend automatisierte Integration wird mithilfe einer maschinenlesbaren Beschreibung des Moduls und seiner Fähigkeiten angestrebt. An der Standardisierung einer solchen Beschreibung arbeiten derzeit verbundene Arbeitskreise der NAMUR, des ZVEI- und der GMA im Rahmen der Initiative „Modular Automation“[1] [7, 8]. Diese herstellerunabhängige Beschreibung wird als *Module Type Package* (*MTP*) bezeichnet. Derzeit beinhaltet das MTP eine Beschreibung der Modulfunktionalitäten, sämtlicher Kommunikationskanäle und -variablen des Moduls sowie aller notwendigen Informationen zur automatisierten HMI-Generierung [8]. Das MTP kann im Idealfall aus einem Engineering-Tool zur Automatisierung des Moduls generiert und z. B. in ein Prozessleitsystem in der PFE importiert werden. Dabei werden nach Auswertung des MTP alle benötigten Treiber, Variablen und HMI-Bilder automatisch im Prozessleitsystem angelegt, womit die informationstechnische Integration der Module in die PFE mit geringem Aufwand vorgenommen werden kann, vgl. [9].

1.2 Kapselung von Modulfunktionalitäten in Services und ihre Abhängigkeiten

Dem MTP-Konzept folgend werden die vom Modul angebotenen verfahrenstechnischen Funktionalitäten in Form von *Diensten* bzw. *Services* gekapselt [10]. Solche Services sind bspw. verfahrenstechnische Grundoperationen wie Rühren oder Filtern oder aber auch Hilfsfunktionen wie Reinigen oder Leeren. Das automatisierungstechnische Integrationsengineering, also die funktionale Integration der Module in die PFE und das logische Verbinden von Services unterschiedlicher Module, setzt eine Serviceorchestrierung und -parametrierung voraus [10, 11]. Daraus ergibt sich aus automatisierungstechnischer Sicht eine Software-Architektur, die stark an die aus der informationstechnischen Domäne bekannte *Microservice-Architektur* angelehnt ist [12]. Eine herstellerübergreifende Integration ist dabei nur möglich, wenn eine vereinheitlichte Ausführung der Services gewährleistet ist. Hier sieht das Konzept der NAMUR eine zustandsbasierte Steuerung von Services angelehnt an die chargenorientierte Fahrweise nach IEC61512/ISA88/PackML [13] vor. Dazu werden Zustände und Zustandsübergänge von Prozedurelementen mithilfe eines Zustandsautomaten beschrieben. Ein an die IEC61512 – edition 2 angelehnter Automat zur zustandsbasierten Steuerung von Services ist in Abb. 1 dargestellt. Jeder verfahrenstechnische Service wird durch einen solchen Zustandsautomaten modelliert und kann die enthaltenen Zustände wie „Idle“, „Running“ oder „Stopped“ einnehmen. Zwei wesentliche Vorteile der Abstraktion von verfahrenstechnischen Funktionen in Form von standardisierten Zustandsautomaten sind die Möglichkeit zur herstellerunabhängigen Aggregation von Modulen zu Produktionssystemen und die Möglichkeit, das Automatisierungsengineering modellbasiert vornehmen zu können [14].

1 Im Folgenden zur besseren Lesbarkeit unter „NAMUR“ zusammengefasst, da von ihr die Initiative ausging.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

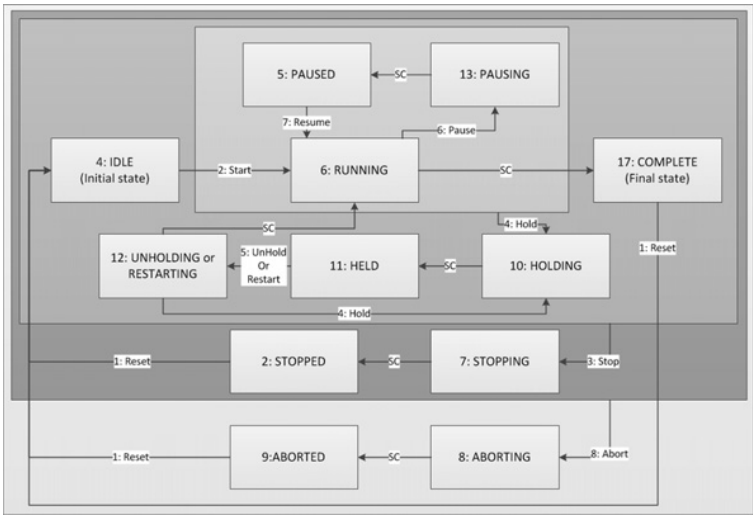


Abb. 1: Service-Zustandsautomat in Anlehnung an IEC 61512 – edition 2 [13].

Noch offen ist in der Diskussion in den o. g. Gremien, ob die Zustände eines Services in nur *einem* Automaten abgebildet werden können. Denn auch in der Welt der modularen Prozessanlagen bleibt der Bedarf an verschiedenen Modul- oder Service-Betriebsarten, wie z. B. Handbetrieb und Automatikbetrieb, bestehen. Folgt man der zustandsbasierten Prozessführung weiter, müssten für jede Betriebsart ein entsprechender Zustandsautomat (oder eine Untermenge dessen) sowie Übergangsbedingungen dazwischen existieren [15, 16].

Im Gegensatz zu informationstechnischen Services (wie Webservices) können verfahrenstechnische Services verschiedene Abhängigkeiten untereinander aufweisen [16]. Diese Abhängigkeiten können modulintern, aber auch modulübergreifend bestehen. Gründe hierfür können verfahrens- oder sicherheitstechnischer Natur sein (z. B. Vermeidung gefährlicher Zustände, modulintern sowie -extern) sowie automatisierungstechnische Hintergründe haben (modulintern, z. B. Verwendung gleicher Assets durch unterschiedliche Services).

Aus Sicht der Automatisierung bedeutet dies, dass Services sich zum einen gegenseitig aufrufen, stoppen, abbrechen o. ä. können sollten. Zum anderen ist das gegenseitige Ausschließen von Services in bestimmten Zuständen notwendig [16].

Ein „Mischen"-Service könnte bspw. einen „Rühren"-Service starten und bei Beendigung des Mischens auch wieder stoppen. Auf der anderen Seite darf ein „Befüllen"-Service nicht gestartet werden, wenn gerade „Entleeren" ausgeführt wird. Unter Berücksichtigung eines Betriebsartenkonzeptes ist der Modulingenieur demnach unter anderem mit folgenden Fragen konfrontiert:

- Welche Betriebsarten unterstützen das Modul und dessen Services?
- In welchen Zuständen dürfen die Betriebsarten der Services gewechselt werden?
- Welche Services schließen sich in welchen Betriebsarten aus?
- Welche Services werden mit welchen weiteren Services synchronisiert oder rufen diese auf?

Es zeichnet sich ab, dass die Summe aller Abhängigkeiten schnell zu einer hohen Komplexität führen kann. Zudem muss berücksichtigt werden, dass der Einsatz des Moduls und die zugehörige Serviceorchestrierung, d. h. die sequentielle und parallele Anordnung von Serviceaufrufen zur Realisierung des Gesamtproduktionsprozesses, in der PFE während des Modulengineerings nicht bekannt ist. Der Modulingenieur hat dennoch unter allen Umständen sicherzustellen, dass eine Kombination von Serviceaufrufen, die einen nicht-sicheren Modulzustand zur Folge haben könnte, von der PFE nicht vorgenommen werden kann.

Daraus folgend müssen Serviceabhängigkeiten zum einen im MTP beschrieben sein (vgl. [7]), damit der Integrationsingenieur bzw. das Integrationstool diese Abhängigkeiten kennt. Zum anderen müssen diese Abhängigkeiten auch im Programmcode der Modulautomatisierung implementiert sein, damit ein fehlerhafter Aufruf den ungewünschten Zustand bzw. Zustandswechsel nicht hervorruft, sondern das Modul den Aufruf zurückweist. Des Weiteren sind eine Evaluierung und insbesondere die Verifikation der Serviceabhängigkeiten zur Überprüfung der Auswirkungen auf den späteren Modulbetrieb notwendig.

Dementsprechend lassen sich die elementaren Fragen zusammenfassend wie folgend formulieren:

- Welche Bedingungen gelten für den Wechsel von Zuständen in Services?

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

- Wie sieht eine möglichst geeignete Implementierung der Serviceabhängigkeiten aus?
- Wie kann sichergestellt werden, dass die Bedingungen und ihre Implementierung logisch konsistent sind und das Modul stets lauffähig ist?

Hieraus resultiert der Bedarf an einer komplexitätsreduzierenden Entwurfs- und Verifikationsmethode für Serviceabhängigkeiten des Moduls. Eine solche Methode wird in diesem Beitrag vorgestellt. Hierbei wird das Engineering mit einer an die *Cause and Effect Matrix* [17] angelehnten *Serviceabhängigkeitsmatrix* (*SA-Matrix*) überschaubar in tabellarischer Form vorgenommen. Eine formale Beschreibung der SA-Matrix erlaubt die Definition von Transformationsregeln, um die Abhängigkeiten in ein Beschreibungsformat zu transformieren, so dass eine automatisierte Überprüfung des Zustandsraumes und somit Verifikation der Serviceabhängigkeiten vorgenommen werden kann. Im speziellen handelt es sich bei dem Beschreibungsmittel um Petri-Netze unter Nutzung des Austauschformats *Petri Net Markup Language* (*PNML*) [18].

Der vorliegende Beitrag ist dabei wie folgt gegliedert: Der folgende Abschnitt beschäftigt sich mit dem Stand von Forschung und Technik für den Entwurf und die Verifikation von Abhängigkeiten innerhalb von Automatisierungssoftware. In Abschnitt 3 wird die vorgeschlagene Methode in Form eines Workflows für den Entwurf und die Verifikation von Serviceabhängigkeiten (SA) vorgestellt. Darauf aufbauend wird in Abschnitt 4 die SA-Matrix sowie ihre formale Beschreibung erläutert. Abschnitt 5 beinhaltet die Darstellung der Serviceabhängigkeiten in PNML und die Transformation der SA-Matrix nach PNML. Der Beitrag schließt mit einer Fallstudie in Abschnitt 6 sowie einer Zusammenfassung und einem Ausblick auf kommende Arbeiten in Abschnitt 7 ab. Die in diesem Beitrag vorgestellten Inhalte wurden von den Autoren im gemeinsamen Forschungsprojekt DIMA (Dezentrale Intelligenz für Modulare Automation) erarbeitet. Weitere Ergebnisse des Projekts DIMA (Laufzeit 2014 bis 2017) wurden in [9, 16] veröffentlicht.

2 Stand der Forschung zu Abhängigkeiten und deren Verifikation

Der Bedarf an der Darstellung und/oder der Analyse von Abhängigkeiten zwischen Services, Applikationen oder Prozessen besteht in unterschiedlichen Domänen. Die entsprechenden Ansätze und Methoden sind somit stets auf ein domänenspezifisches Ziel ausgerichtet. Im Folgenden werden daher Ansätze aus unterschiedlichen Domänen vorgestellt und hinsichtlich ihrer Tauglichkeit zur Anwendung für die in Abschnitt 1 beschriebene Aufgabe diskutiert.

Geschäftsprozesse werden häufig als Services verstanden und zeigen auch viele Abhängigkeiten untereinander auf. Nicht selten werden Ergebnisse in Form von Dokumenten und Daten anderer Prozesse zur Durch- oder Weiterführung eines Geschäftsprozesses benötigt. Reisig et al. [19] zeigen beispielsweise einen Ansatz zur Modellierung von Geschäftsprozessen als Services sowie deren Abhängigkeiten in Form von Kommunikationsbeziehungen auf. Hierzu nutzen sie die *Process Execution Language for Web Services* (*BPEL*), eine ausführbare Sprache zur Darstellung von kommunizierenden Geschäftsprozessen. Für BPEL stellen Reisig et al. eine Petri-Netz Semantik vor, um sie zu formalisieren und eine graphische Darstellung von BPEL zu ermöglichen. Allerdings herrschen in der modularen Prozessautomation andere Beziehungen zwischen Services vor als sie mit BPEL beschrieben werden können. Zudem ist durch den Ansatz keine komplexitätsreduzierende Methode zum Entwurf dieser Beziehungen gegeben.

Einen weiteren Ansatz aus dem Bereich der Geschäftsprozessmodellierung liefern Buchwald und Bauer in [20]. Insbesondere beschreiben sie die Aufrufbeziehungen zwischen Applikationen und Geschäftsprozessen. Die Autoren argumentieren, dass die BPEL-basierte Service-Orchestrierung eine häufig für Modellierer unverständliche Sicht aufweist. Daher, und aufgrund der Vielzahl an Grundobjekten in der BPEL, haben sie sich für *ereignisgesteuerte Prozessketten* (*EPK*) als Beschreibungsmittel entschieden. Als Alternative zu diesen nennen sie *UML-Aktivitätsdiagramme*. Eine automatisierte Verifikation der Beziehungen wird hier nicht vorgenommen. Hierfür fehlt den genannten Beschreibungsmitteln der nötige Formalisierungsgrad.

Dieser wird bspw. in [21] durch die Nutzung von Prädikatenlogik geliefert. Prädikatenlogik wird hier zur Beschreibung von Regeln genutzt, welche die Abhängigkeiten zwischen Produkten und Serviceleistungen wiedergeben. Aufgrund des hohen Formalisierungsgrades steht eine Vielzahl an automatischen Analysen zur Verfügung. Allerdings gestaltet sich die Modellierung in Prädikatenlogik als kompliziert. Sie eignet sich daher nicht zur Komplexitätsreduzierung bei der Beschreibung von Serviceabhängigkeiten.

Einen Ansatz, der die Modellierung von Prozessen und Web-Services vereint, diskutieren Heinrich et al. in

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

[22]. Speziell stellen sie ein integriertes Konzept zur automatisierten Modellierung und Ausführung von Prozessen unter Verwendung von Web-Services vor. Hierbei werden die Abhängigkeiten zwischen Aktionen ermittelt und in einem *Aktionsabhängigkeitsgraphen* dargestellt. Aus diesem wird dann ein Aktions-Zustands-Graph gebildet, der alle möglichen Ablaufreihenfolgen der Aktionen enthält. Der Bekanntheitsgrad von Aktionsabhängigkeitsgraphen in der Prozessindustrie ist eher gering, daher eignet sich dieser Ansatz nicht gut für die vorgenannte Aufgabe.

Im Bereich der Service-orientierten Architekturen (SOA) sieht das OASIS-Referenzmodell die Darstellung von Serviceabhängigkeiten innerhalb des *Process Models* vor [23]. Wie diese beschrieben werden, lässt das Referenzmodell jedoch offen. Es existieren in den Domänen Service-orientierter sowie komponentenbasierter Software verschiedene Ansätze, die sich mit der Modellierung von Serviceabhängigkeiten beschäftigen. Cervantes und Hall [24] bspw. liefern einen Ansatz aus dem Bereich Komponenten-basierter verteilter Software-Architekturen. Das zugrundeliegende Paradigma ist das *Service oriented Programming* (*SOP*). Zwei Arten von Abhängigkeiten werden in [24] berücksichtigt. *Component-to-Service* Abhängigkeiten bedeuten, dass eine Komponente von einem Service abhängt, ohne selbst einen Service anzubieten. Bietet eine Komponente Services an, benötigt jedoch andere Services hierzu, wird dies als S*ervice-to-Service* Abhängigkeit beschrieben. In einem sog. *Instance Descriptor* werden eine zu instanziierende Komponente sowie die in ihr enthaltenen Services und ihre Abhängigkeiten im XML-Format beschrieben. Dies erlaubt ein automatisiertes und dynamisches Abhängigkeits-Management zur Laufzeit. Aufrufbeziehungen lassen sich dann zwar maschinenlesbar beschreiben, jedoch in für den Menschen nur aufwändig interpretierbarer XML-Notation.

Ebenfalls ein Ansatz, der sich mit der Modellierung von Abhängigkeiten in komponentenbasierten Systemen beschäftigt, ist die Verwendung einer Abhängigkeitsmatrix [25]. Berücksichtigt werden in [25] acht verschiedene Abhängigkeiten. Diese Abhängigkeiten werden in einer Komponenten-Abhängigkeitsmatrix modelliert. Besteht eine Abhängigkeit, so ist dies mit einem entsprechenden Eintrag in der Matrix gekennzeichnet. Die Darstellung in Form einer Matrix macht das Definieren bzw. Anlegen der Abhängigkeiten sehr komfortabel, auch wenn die hier beschriebene Matrix einem proprietären Beschreibungsmittel entspricht. Aufgrund der ebenfalls proprietären Ausrichtung auf komponentenbasierte Software ist dieser Ansatz allerdings nicht für die Prozessautomation zu übernehmen.

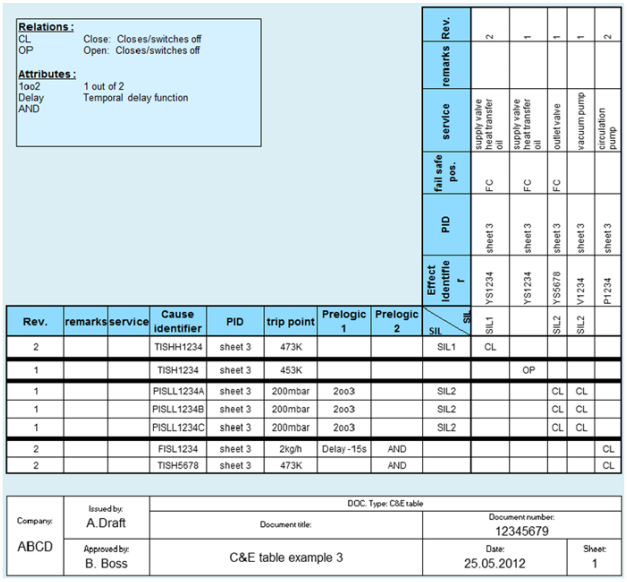
Allgemein scheinen die o. g. Beschreibungsmittel für den Modellierer von Software oder Geschäftsprozessen geeignet, da sie meist in der Domäne gängigen Notationen entsprechen. Automatisierungstechnikern sind diese Beschreibungsmittel jedoch weniger geläufig. Vielmehr sollte im Falle der modularen Prozessautomation ein Beschreibungsmittel gefunden werden, dass sich an in der Automatisierungstechnik bekannten Darstellungsformen orientiert und dennoch den spezifischen Anforderungen gerecht wird. Insbesondere sollte sie vereinbar sein mit der im vorigen Abschnitt aufgezeigten Beschreibung von Service-Zuständen und -Zustandsübergängen.

Ansätze aus dem Bereich der Automatisierung bedienen sich daher häufig der *Unified Modelling Language* (*UML*). Dubnin et al. [26] stellen bspw. einen Engineering-Ansatz für komponentenbasierte verteilte Automatisierungssoftware vor. Hierzu kombinieren sie verschiedene UML-Diagramme mit Funktionsblöcken nach IEC 61499. So ist es möglich, Komponenten hierarchisch und nichthierarchisch in Beziehung zu setzen, indem Relationen des Klassendiagrammes genutzt werden. Aus den UML-Diagrammen werden dann die IEC61499 Funktionsblöcke und ihre Interrelationen generiert. Der Ansatz abstrahiert im Endeffekt IEC61499-Programme in UML-Diagrammen. Allerdings reduziert er nicht die Komplexität des Engineerings, sondern bietet eine andere Darstellung der Abhängigkeiten an. Zudem ist der Ansatz nur für 61499-Applikationen und -Laufzeitsysteme vorgesehen. Eine Verifikation der modellierten Abhängigkeiten sieht der Ansatz nicht vor.

Einen weiteren Ansatz, hier aus dem Bereich der modellbasierten Entwicklung eingebetteter Systeme unter Nutzung unterschiedlicher Beschreibungssprachen, stellen Reichmann et al. [27] vor. Dabei wird eine allgemeine Modellkopplung betrachtet, bei der sog. Kopplungsmodelle verwendet. Konkret werden die Kopplungen mit UML-Beziehungen wie Assoziationen oder Abhängigkeiten (UML-Dependencies) dargestellt. Der Vorteil liegt laut den Autoren zum einen in der Möglichkeit, den zugehörigen Quellcode, der die Kopplung realisiert, automatisch zu generieren (Modell-zu-Code Generierung). Zum anderen kann das Kopplungsmodell vom Entwickler und per *Model-Checking* analysiert und verifiziert werden. Dies entspricht exakt der Aufgabe, bei der auch im Rahmen dieses Beitrages der Modulingenieur unterstützt werden soll. Eine konkrete Methode hierzu liefern Reichmann et al. allerdings nicht. Zudem bleiben die Autoren eine Formalisierung, die zur Code-Generierung und auch zur formalen Analyse benötigt wird, schuldig.

Eine proprietäre Erweiterung der UML nutzen Fengler et al. in [28]. Speziell erweitern sie die Semantiken von

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43



Relations :
CL Close: Closes/switches off
OP Open: Closes/switches off

Attributes :
1oo2 1 out of 2
Delay Temporal delay function
AND

Rev.	remarks	service	Cause identifier	PID	trip point	Prelogic 1	Prelogic 2	SIL \ Effect identifier	SIL1		SIL2	SIL2	
								PID	sheet 3	sheet 3	sheet 3	sheet 3	sheet 3
								fail safe pos.	FC	FC	FC		
								service	supply valve heat transfer oil	supply valve heat transfer oil	outlet valve	vacuum pump	circulation pump
								remarks					
								Rev.	2	1	1	1	2
2			TISHH1234	sheet 3	473K			SIL1	CL				
1			TISH1234	sheet 3	453K					OP			
1			PISLL1234A	sheet 3	200mbar	2oo3		SIL2			CL	CL	
1			PISLL1234B	sheet 3	200mbar	2oo3		SIL2			CL	CL	
1			PISLL1234C	sheet 3	200mbar	2oo3		SIL2			CL	CL	
2			FISL1234	sheet 3	2kg/h	Delay -15s	AND						CL
2			TISH5678	sheet 3	473K		AND						CL

Company:	Issued by:	DOC. Type: C&E table	
	A.Draft	Document title:	Document number: 12345679
ABCD	Approved by: B. Boss	C&E table example 3	Date: 25.05.2012 / Sheet: 1

Abb. 2: Cause and Effect Matrix nach IEC 62881 [17].

UML-Statecharts und nennen diese *colored state charts*. Diese kombinieren state charts als einfache Modellierungssprache für den Ingenieur und High-Level-Petri-Netze mit ihrer Ausdrucksstärke. Eine Transformation in farbige Petri-Netze erlaubt die formale Verifikation der Abhängigkeiten. Jedes Teilverhalten wird dabei mittels einer Farbe modelliert. Das resultierende Gesamtnetz unter Berücksichtigung aller Farben beschreibt somit das abhängige Verhalten. Die Autoren postulieren, dass dies die Modellierung von verteilten Anwendungen und ihrer Abhängigkeiten übersichtlich gestaltet und mithilfe der Transformation in Petri-Netze eine Vielzahl an bestehenden formalen Verifikationsmethoden zulässt. Eine explizite Kopplung verschiedener Teil-Netze wird hier jedoch nicht vorgenommen.

Ein speziell in der Prozessautomation verwendetes Beschreibungsmittel zum Engineering von Abhängigkeiten ist die *Cause & Effect Matrix* (*C&E Matrix*) [17]. Die C&E Matrix dient der Darstellung von nötigen Aktoreingriffen (Effects) in Abhängigkeit von Prozesszuständen (Causes). So kann bspw. dargestellt werden, dass Ventile geschlossen oder geöffnet werden müssen, wenn bestimmte Füllstände oder Drücke erreicht wurden o. ä., vgl. Abb. 2. Ein wesentlicher Vorteil liegt darin, dass C&E Matrizen ein bereits bekanntes Mittel für den Entwurf und die Implementierung von Abhängigkeiten in der Prozessautomation sind. So befinden sich diese nicht nur derzeit in der internationalen Standardisierung (DIN EN62881 [17]), sondern werden bereits auch in Automatisierungs-Werkzeugen für die Implementierung von Feldgerät-Abhängigkeiten eingesetzt.[2] Ein Ansatz, der es erlaubt, C&E Spezifikationen sowie deren Implementierung zu testen, wird in [29] präsentiert. Hierzu wird die Spezifikation in ein Petri-Netz transformiert, auf dessen Basis sie getestet werden kann. Des Weiteren werden aus dem Petri-Netz automatisch Testfälle generiert. Diese Testfälle werden genutzt, um die Implementierung der C&E Spezifikation zu testen. Der Ansatz würde sich prinzipiell auch für die hier vorgestellte Problemstellung eignen. Allerdings sind C&E Matrizen, wie sie typischerweise in der Prozessindustrie eingesetzt und auch in der DIN EN62881 standardisiert werden, nicht geeignet, alle Abhängigkeiten abzubilden, die zwischen verfahrenstechnischen Services eines Moduls vorherrschen können. So sehen diese C&E Matrizen z. B. vor, dass stets eine Ursache eine aktive Wirkung zur Folge hat. Beziehungen wie ein gegenseitiger Ausschluss lassen sich so jedoch nicht darstellen, da die C&E Matrix ausschließlich zur Darstellung Ursache-Wirkungs-Beziehungen entwickelt wurde.

Da C&E Matrizen jedoch ein bekanntes Mittel für das Engineering von Abhängigkeiten sind und auch die Möglichkeit zur Transformation in formale Beschreibungsmittel erlauben, verfolgt der hier dargestellte Ansatz eine ähnliche Methodik. Diese wird im folgenden Abschnitt vorgestellt.

3 Workflow zum Engineering und zur Verifikation von Serviceabhängigkeiten

Ziel des hier vorgestellten Ansatzes ist es, den Modulingenieur beim Engineering der Serviceabhängigkeiten zu unterstützen, indem ihm eine Methode zur Verfügung gestellt wird, die es erlaubt, die Abhängigkeiten übersichtlich darzustellen, ohne dabei funktionale Einschränkungen hinnehmen zu müssen. Zudem ist der Ansatz auf eine unkomplizierte Umsetzung der Methode in einem Softwarewerkzeug ausgerichtet. Abb. 3 zeigt die Vorgehensweise zum Entwurf und zur Verifikation von Serviceabhängigkeiten in einem Workflow.

Die initial zugrundeliegende Information stellt hier die Deklaration aller Services und ihrer Betriebsarten dar. Da diese Information offensichtlich modulspezifisch ist, muss sie vom Modulingenieur selbst angegeben werden.

2 Siehe z. B: http://w3.siemens.com/mcms/process-control-systems/de/simatic-pcs-7/simatic-pcs-7-systemkomponenten/engineering-system/pages/logic-matrix.aspx

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

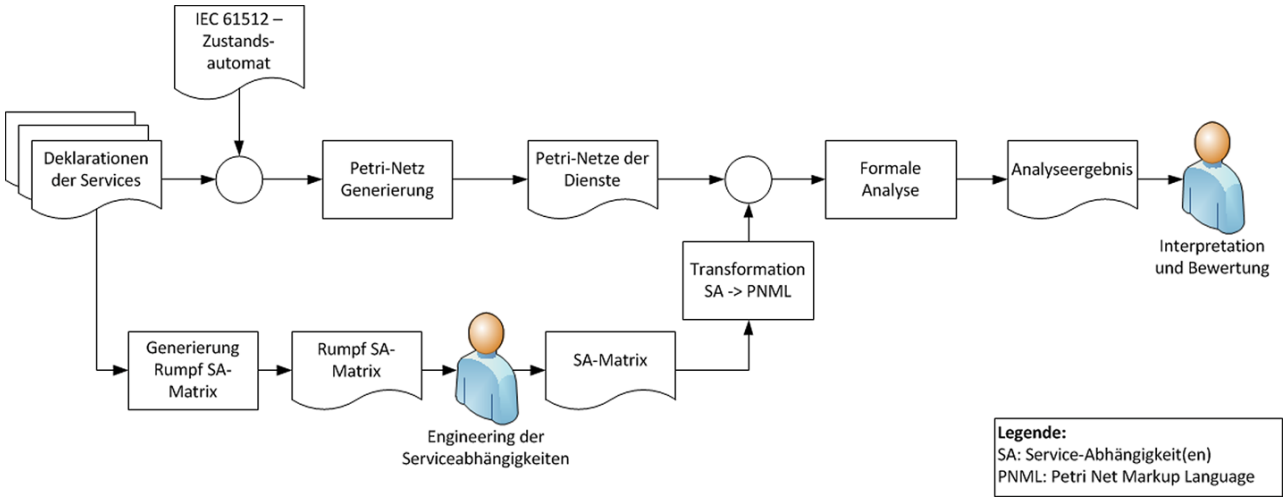


Abb. 3: Allgemeiner Workflow zum Entwurf und zur Verifikation von Serviceabhängigkeiten.

Aus den Deklarationen kann nun zum einen der Rumpf für eine an die C&E Matrix angelehnte Serviceabhängigkeitsmatrix generiert werden. Sie dient dem Ingenieur zum Entwurf der Abhängigkeiten in einer übersichtlichen tabellarischen Darstellung. Zum anderen wird für jeden Service in jeder Betriebsart ein Petri-Netz generiert, das den Zustandsautomaten gem. IEC61512 edition 2 repräsentiert. Da der Zustandsautomat zur Repräsentation eines Services standardisiert ist (bzw. sein wird), ist seine Struktur bekannt, und es kann ohne zusätzlichen Aufwand jeder Service in ein Petri-Netz überführt werden. Durch die Verwendung von Petri-Netzen steht ein mathematisch-formales Beschreibungsmittel zur Verfügung, das es erlaubt, die Nebenläufigkeit verschiedener Services abzubilden und das einer automatischen Analyse unterzogen werden kann. Zudem bietet die PNML eine standardisierte, XML-basierte und somit maschinenlesbare Sprache zur Beschreibung der Petri-Netze. Es sei erwähnt, dass die generierten Petri-Netz-Strukturen dem Modulingenieur nicht präsentiert werden, da für ihn ausschließlich die Service-Abhängigkeits-Matrix (SA-Matrix) und die Analyseergebnisse relevant sind.

Sind die gewünschten Abhängigkeiten in die SA-Matrix eingetragen worden, so wird diese mittels Modell-zu-Modell-Transformation in die Petri-Netze überführt. Für eine solche Transformation sind Meta-Modelle der beiden Modellarten notwendig [30]. Die ISO/IEC15909-2 [18] liefert ein solches Meta-Modell für PNML, was einen weiteren Vorteil des gewählten Ansatzes darstellt. Ein Meta-Modell für die SA-Matrix wird im folgenden Abschnitt vorgestellt. Nachdem die Abhängigkeiten in das gesamthafte Petri-Netz überführt wurden, kann dieses hinsichtlich formaler Eigenschaften analysiert und verifiziert werden. Die Analyseergebnisse werden dem Modulingenieur zur Verfügung gestellt, damit dieser die Ergebnisse interpretieren und bewerten kann.

4 Engineering von Serviceabhängigkeiten mithilfe einer Serviceabhängigkeitsmatrix

Die SA-Matrix dient dem Modulingenieur zum Formulieren der Serviceabhängigkeiten während des projektunabhängigen Modulengineerings. Somit sollte die Matrix möglichst leicht verständlich sein und dennoch die nötige Flexibilität aufweisen, um alle denkbaren Abhängigkeiten zwischen verfahrenstechnischen Services innerhalb eines Moduls modellieren zu können. Alle Elemente der Matrix müssen eindeutig beschrieben sein. Dies ist zum einen notwendig, um eine SA-Matrix zur Analyse der Abhängigkeiten in das Petri-Netz-Konstrukt zu überführen. Zum anderen erlaubt diese Eindeutigkeit eine Quellcode-Generierung zur Abbildung der Abhängigkeiten in SPS-Steuerungen nach IEC 61131-3 [31].[3] Ein solcher Formalisierungsgrad ergibt sich aus der Beschreibung mithilfe eines Meta-Modells [30].

Das Meta-Modell der SA-Matrix zeigt Abb. 4. Dieses beschreibt die SA-Matrix, bestehend aus einer Menge von Abhängigkeitsquellen (*Source*) und Abhängigkeitszielen

3 Siehe z. B. [32] zur Quellcode-Generierung aus C&E-Matrizen.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

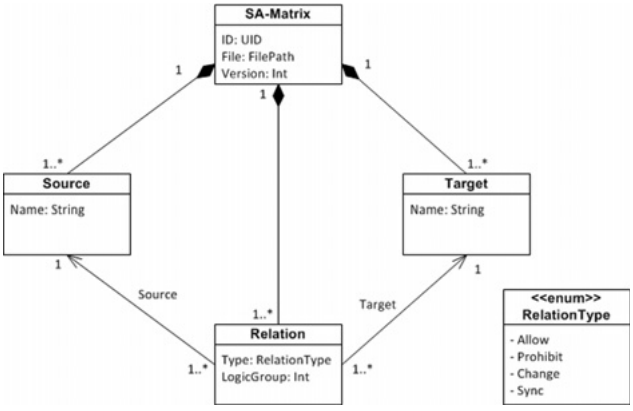


Abb. 4: Meta-Modell der Serviceabhängigkeitsmatrix.

(*Target*). Diese werden im Folgenden schlicht als Quelle und Ziel bezeichnet. Zwischen Quellen und Zielen können Beziehungen (*Relations*) bestehen, welche die Abhängigkeit beschreiben. Zur eindeutigen Identifizierung ist jede SA-Matrix mit einer Identifikationsnummer (ID) und einer Versionsnummer versehen. Da die Bearbeitung in einer Datei, z. B. einer Excel-Datei, stattfinden soll, verweist ein SA-Matrix-Objekt mithilfe des *File*-Attributes auf diese Datei.

Sowohl eine Quelle als auch ein Ziel kann entweder ein Zustand (d. h. ein Zustand des in Abb. 1 dargestellten Automaten) oder ein Zustandsübergang (d. h. eine Kante des in Abb. 1 dargestellten Automaten) eines Services sein. Dies hängt von der Art der Relation ab. Die Art der Relation wird über das Attribut *Type* beschrieben und entspricht einem der folgenden vier Typen:

1. **Allow:** Eine Allow-Relation hat stets einen Zustand eines Services als Quelle und einen Zustandsübergang eines anderen Services als Ziel. Sie ist so zu verstehen, dass der Ziel-Zustandsübergang nur vorgenommen werden darf, wenn der Quell-Zustand derzeit aktiv ist. So kann bspw. modelliert werden, dass ein Service nur gestartet werden darf, wenn sich ein anderer Service in einem bestimmten Zustand (z. B. Idle) befindet.
2. **Prohibit:** Wie die Allow-Relation, hat die Prohibit-Relation stets einen Zustand eines Services als Quelle und einen Zustandsübergang eines anderen Services als Ziel. Allerdings bedeutet diese Relation, dass der Ziel-Übergang nur stattfinden darf, wenn der Quell-Zustand derzeit nicht aktiv ist. Der Zustandsübergang des Ziels ist bei aktivem Zustand der Quelle also verboten. Diese Relation dient bspw. der Abbildung, dass ein bestimmter Service nicht gestartet werden darf, wenn ein anderer Service sich im Zustand Running befindet.
3. **Change:** Die Change-Relation dient der Abbildung eines Betriebsartenwechsels. Entsprechend muss die Relation bewirken, dass ein Zustandswechsel von einem Zustand des Automaten, der den Service in der Quell-Betriebsart repräsentiert, zu einem anderen Zustand des Automaten, der den Service in der Ziel-Betriebsart repräsentiert, stattfindet. Somit ist ihre Quelle stets ein Zustand eines Services in einer bestimmten Betriebsart und ihr Ziel ein Zustand desselben Services in einer anderen Betriebsart. Mit ihr kann bspw. angegeben werden, dass ein Betriebsartenwechsel vom Automatikbetrieb in den Semi-Automatikbetrieb nur stattfinden darf, wenn sich der Service derzeit im Zustand *Idle* befindet und der Betriebsartenwechsel auch in den Zustand *Idle* des Semi-Automatikbetriebes führt.
4. **Sync:** Die Sync-Relation dient der Synchronisierung von Zustandswechseln zwischen Services. Somit ist ihre Quelle stets ein Zustandsübergang eines Services und ihr Ziel ein Zustandsübergang eines anderen Services. Die Bedeutung einer solchen Relation ist die folgende: Soll der Quell-Zustandsübergang vorgenommen werden, so wird der Ziel-Zustandsübergang auch vorgenommen. Dies beinhaltet konsequenterweise implizit die Bedingung, dass der Ziel-Zustandsübergang auch vorgenommen werden kann, also der zugehörige Service sich in einem entsprechenden Zustand befindet. So kann bspw. abgebildet werden, dass ein Service beim eigenen Starten ebenfalls einen anderen Service startet. Somit entspricht eine Sync-Relation einem Serviceaufruf bzw. dem Senden eines Befehls zum Zustandswechsel. Die Sync-Relation kann aber auch genutzt werden, um bspw. alle Services abzubrechen, wenn ein Service abgebrochen wird.

Da die Bedingung für einen Zustandswechsel auch von mehreren Quellen abhängig sein kann, muss es zudem möglich sein, Relationen mehrerer Quellen zu einem Ziel zu definieren und diese logisch miteinander zu verknüpfen. Die Startbedingung für einen Service kann bspw. von aktiven Zuständen mehrerer Services (z. B. Service darf nur starten, wenn alle anderen Services in Idle sind) oder von inaktiven Zuständen mehrerer Services (z. B. Service darf nur starten, wenn kein anderer Service in Zustand Running) abhängig sein. Denkbar sind auch ODER-Verknüpfungen von Bedingungen (z. B. Service A darf nur starten wenn Service B oder Service C (nicht) in Running ist). Oder eben Kombinationen von UND- und ODER-Bedingungen. Um dies in der SA-Matrix abbilden zu können, werden die Relationen über das Attribut *LogicGroup*

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

Tab. 1: Beispiel einer SA-Matrix.

Quelle	Ziel *C.Starting*
A.Idle	Allow 1
B.Idle	Allow 1
C.Running	Prohibit 2

in logische Gruppen unterteilt. Alle Relationen eines Ziels, die sich in derselben Gruppe befinden sind dabei UND-verknüpft zu interpretieren und alle Gruppen untereinander sind ODER-verknüpft zu interpretieren. Es entsteht also eine disjunktive Normalform der Relationen. Folgendes Beispiel soll dies verdeutlichen:

Ein Zustandsübergang *X.Starting* soll nur stattfinden dürfen, wenn entweder die beiden Zustände *A.Idle* und *B.Idle* aktiv sind oder *C.Running* nicht aktiv ist. In diesem Fall würden die beiden Zustände *A.Idle* und *B.Idle* einer logischen Gruppe zugeordnet werden und *C.Running* einer anderen logischen Gruppe. Teilt man entsprechend erstere beiden Abhängigkeiten in die logische Gruppe 1 ein und letztere in die Gruppe 2, ergibt sich die in Tabelle 1 dargestellte SA-Matrix.

Es sei hier zu erwähnen, dass logische Verknüpfungen von Change-Relationen sowie eine UND-Verknüpfung von Sync-Relationen nicht sinnvoll sind und daher hier auch nicht berücksichtigt werden.

5 Modellierung der Services und ihrer Abhängigkeiten mithilfe der Petri Net Markup Language

In diesem Abschnitt wird nun beschrieben, wie die in der SA-Matrix beschriebenen Abhängigkeiten in einem Petri-Netz-Konstrukt dargestellt werden können. Hierzu ist es zunächst notwendig, den in Abb. 1 dargestellten Automaten als Petri-Netz in PNML zu überführen. Für eine übersichtliche Darstellung in XML wurde hierzu ein hierauf zugeschnittener Petri-Netz-Typ definiert, die *ServiceNets*. Dieser baut auf dem standardisierten PNML-Kernmodell auf.

Im folgenden Abschnitt werden daher zunächst Grundlagen zur PNML vorgestellt und darauf aufbauend die Definition der ServiceNets sowie die Darstellung des Service-Zustandsautomaten in ServiceNets. Abschnitt 5.4 stellt die Modellierung der Abhängigkeiten in ServiceNets vor.

5.1 Die Petri Net Markup Language

Die PNML ist Inhalt der ISO/IEC 15909-2 [18] und beschreibt eine XML-basierte Syntax für Petri-Netze im Allgemeinen und die Semantik von *High-Level-Petri-Netzen* im Speziellen. Die Modellierung ist so generisch, dass sie die Beschreibung weiterer Petri-Netz-Typen zulassen, wie in der Norm auch bspw. für *Stellen-Transitions-Netze* vorgenommen. Das Definieren weiterer Petri-Netz-Typen ist dabei durchaus gewollt. In PNML werden Petri-Netze allgemein als gerichtete Graphen beschrieben. Alle spezifischen Informationen werden als Labels hinzugefügt.

PNML wird in einem Meta-Modell, dem sog. Kernmodell, beschrieben. Dieses besteht aus zwei Teilen: a) Konzepte und b) graphische Informationen. Im Rahmen der Arbeiten dieses Beitrages ist ausschließlich der Teil der Konzepte relevant. Die wesentlichen Aspekte des zugehörigen Kernmodells sind in Abb. 5 dargestellt.

Ein PNML-Dokument wird *Petri Net Document* (*PetriNetDoc*) genannt, wenn es dem Kernmodell entspricht. Es enthält eines oder mehrere Petri-Netze (*PetriNet*). Jedes Petri-Netz entspricht einem Typen und besitzt einen unique identifier (UID). Der Typ des Netzes (bspw. P/T[4] oder *symmetric net*, siehe [18]) referenziert auf das Dokument, das diesen Typen definiert.

Ein PetriNet besteht aus einer oder mehreren *Pages*, die *Objekte* (*Object*) beinhalten. Objekte besitzen eine UID und beschreiben die Struktur des Netzes. Die wichtigsten Objekte sind Seiten (*Pages*), Stellen (*Places*), Transitionen (*Transitions*) und gerichtete Kanten (*Arcs*). Allgemein wird von Knoten (*Node*) und Kanten gesprochen.

Seiten können weitere Seiten enthalten. Kanten dürfen nur innerhalb einer Seite existieren, dürfen also nicht Knoten seitenübergreifend verbinden. Um dennoch zwischen Stellen und Transitionen unterschiedlicher Seiten verweisen zu können, existieren Referenzknoten. Ein Referenzknoten referenziert auf eine Stelle (*RefPlace*) oder auf eine Transition (*RefTransition*) und entspricht somit einer Repräsentation eines Knoten auf einer anderen Seite [18].

Labels dienen der Beschreibung weiterer Bedeutung von Objekten. Typische Labels sind (ohne darauf beschränkt zu sein):

4 Place/Transition.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

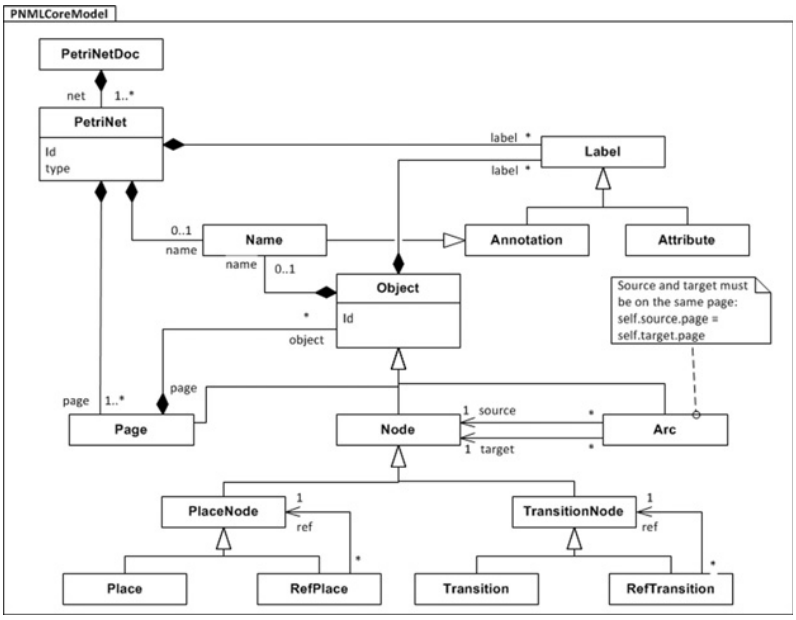


Abb. 5: Hier wesentliche Aspekte des PNML-Kernmodells: Aspekte nach ISO/IEC 15909-2 [18].

- Der Name eines Knoten
- Eine Transitionsbedingung
- Eine Kantenannotation, z. B. Gewichtung
- Verschiedene globale Labels wie Deklarationen von Variablen oder Operatoren.

Labels werden in Annotationen (*Annotations*) und Attribute (*Attributes*) unterschieden. Annotationen tauchen typischerweise als Text neben dem entsprechenden Objekt auf und beschreiben z. B. die Darstellungsform des Objektes. Das einzige von PNML vordefinierte Label ist die Annotation des Namens (eines Objektes oder Petri-Netzes).

5.2 Definition des Petri-Netz-Typen ServiceNets

Um das Modell zur Darstellung von Serviceabhängigkeiten möglichst kompakt und die Auswertung effizient zu halten, wird hier ein neuer Petri-Netz-Typ (*ServiceNets*) auf Grundlage des PNML-Kernmodells vorgestellt. Dieser entspricht in großen Teilen Bedingungs-Ereignis-Netzen (B/E-Netz) mit einigen Erweiterungen, die für eine kompakte Darstellung hilfreich sind. Dies hat den Vorteil, dass vorhandene Verifikationsmechanismen verwendet werden können und eine Transformation in B/E-Netze zu Analysezwecken möglich ist. Das Meta-Model für ServiceNets als Erweiterung des PNML-Kernmodells ist in Abb. 6 dargestellt. Alle Elemente des Kernmodells bleiben dabei erhalten und werden lediglich um die in Abb. 6 gezeigten ergänzt.

Folgende Erweiterungen sind Teil der ServiceNets:

- *ServiceNet* als neuer PetriNetz-Typ (erbt von *PetriNet-Type* aus dem PNML-Kernmodell)
- *Marking* als Attribut einer Stelle. Enthält einen booleschen Wert, der die Markierung angibt, wie bei B/E-Netzen.
- *ArcType*: Ein Attribut von *Arc*, der dessen Typ angibt. Möglich sind:
  - *normal*: Eine Kante mit diesem Typen entspricht einer Kante, wie sie aus B/E-Netzen bekannt ist
  - *read:* Entspricht einer Testkante, vgl. [33]
  - *inhibitor*: Entspricht einer Inhibitorkante, vgl. [33]
  - *reset*: Diese Kante kann zwischen einer Seite und einer Transition existieren. Sie ist stets aktiviert, wenn mindestens eine Marke in einer Stelle innerhalb der Seite oder einer ihr untergeordneten Seite vorhanden ist. Sie entzieht allen der Quell-Seite unterlagerten Stellen die Marken, wenn ihre Transitionsbedingung TRUE ist (d. h. die annotierte AccessVariable ist TRUE).
- *RefPage*: Eine Referenz auf eine Seite, vergleichbar mit RefTransition und RefPlace des PNML-Kernmodells.

5.3 Umsetzung des Service-Automaten als ServiceNet

Im Folgenden wird dargelegt, wie der in Abschnitt 1.2 vorgestellte Zustandsautomat (Abb. 1) mittels der ServiceNets beschrieben werden kann.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

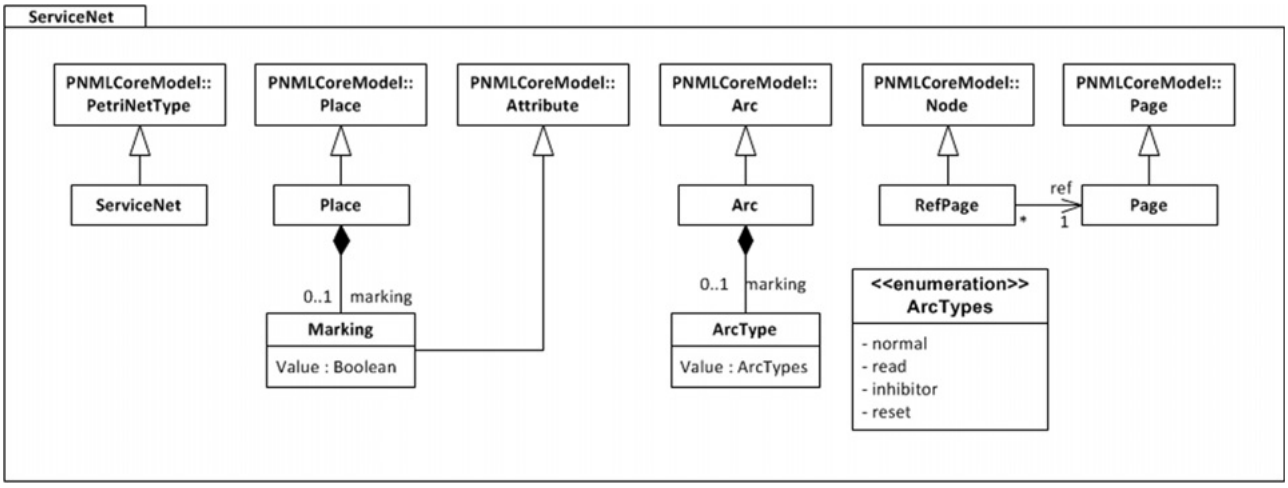


Abb. 6: Meta-Modell der ServiceNets als Erweiterung des PNML-Kernmodells.

Jeder Zustand des Automaten ist als eine Stelle des ServiceNets dargestellt. Ist eine Stelle mit einer Marke besetzt, so bedeutet dies, dass der entsprechende Zustand aktiv ist. Die Zustandsübergänge entsprechen ServiceNet-Transitionen. Bereiche, die jederzeit mittels einer Transition verlassen werden können, werden in einer Seite modelliert. Die Transition, die das Verlassen des Bereiches angibt (z. B. „8: Abort“ in Abb. 7) wird mittels einer von der entsprechenden Page kommenden Reset-Kante modelliert.

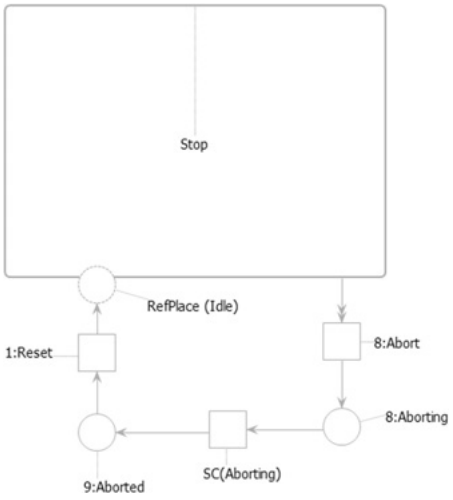


Abb. 7: Seite der höchsten Hierarchiestufe des Service-Automaten als Petri-Netz.

Als Beispiel sei auf Abb. 7 verwiesen. Die dargestellte Seite entspricht der höchsten Hierarchiestufe im Netz, d. h. alle anderen Seiten sind ihr untergeordnet. Sie entspricht dem hellgrauen (unteren) Teil des in Abb. 1 dargestellten Automaten. Die nächstniedrigere Hierarchiestufe stellt dabei die Seite „Stop“ dar (dargestellt durch ein abgerundetes Rechteck). Von ihr ausgehend befindet sich eine Reset-Kante (Pfeil mit Doppelspitze), die auf die Transition „8: Abort“ gerichtet ist.

Da PNML keine direkte Verbindung zwischen Objekten verschiedener Seiten vorsieht, sind die notwendigen Verbindungen mittels Referenz-Transitionen und -Stellen realisiert. Wann immer von einer Seite auf eine Transition oder Stelle einer anderen Seite referenziert werden muss, werden in beiden Seiten entsprechende Referenzen erstellt. (Dies wird stets dargestellt als Stelle/Transition mit gestricheltem Umriss). Prinzipiell würde eine Referenz auf einer Stelle genügen, jedoch macht dieses Vorgehen es Algorithmen leichter, Zusammenhänge zu finden, da die Verbindung bidirektional durch das Modell verfolgt werden kann. Folgendes Beispiel soll dies verdeutlichen:

Abb. 8 zeigt einen weiteren Teil des Zustandsautomaten als ServiceNet. Dargestellt ist die Seite, welche die Zustände *Idle*, *Complete*, *Holding*, *Held* sowie *UnholdingOrRestarting* enthält. Da vom Zustand *Aborted* auf oberster Hierarchieebene (siehe Abb. 7) in den Zustand *Idle* übergegangen werden muss, befindet sich auf der Seite in Abb. 8 eine Referenzstelle, die auf *Idle* verweist. Um nun auch auf der anderen Seite (Abb. 7) diesen Zusammenhang zu erkennen, befindet sich dort eine Referenz-Transition, die auf die Transition *Reset* verweist. Da zusätzlich noch von einer anderen Seite auf *Idle* verwiesen werden muss, befindet sich noch eine zweite Referenztransition in der in Abb. 8 dargestellten Seite.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

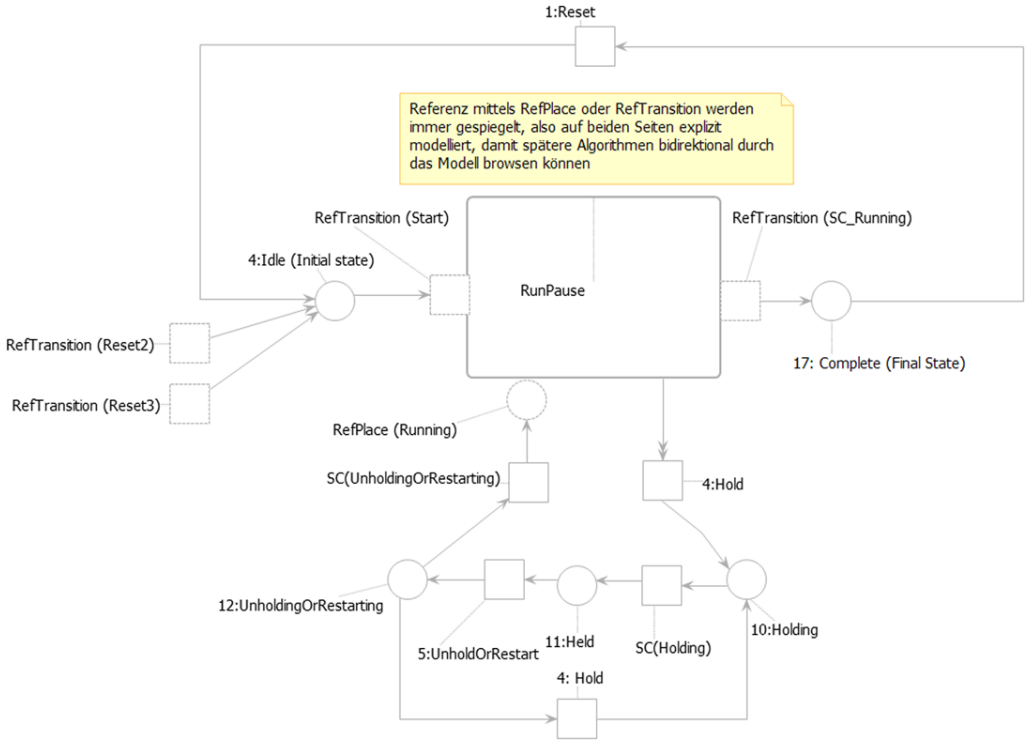


Abb. 8: Teil des Service-Zustandsautomaten als ServiceNet.

5.4 Darstellung der Abhängigkeiten in ServiceNets

Entsprechend des in Abb. 3 dargestellten Workflows wird für jeden Service und jede Betriebsart ein Service-Automat als ServiceNet erzeugt. Hierzu wird für jeden Service eine Seite angelegt und innerhalb dieser Seite wiederum je eine Seite für jede Betriebsart. Hierin werden Kopien der in Abschnitt 5.3 beschriebenen ServiceNet-Repräsentation des Service-Automaten angelegt. Für jeden Service kann stets nur eine Betriebsart aktiv sein. Entsprechend darf nur exakt eine Marke in der Initialmarkierung eines Services existieren. Um die Serviceabhängigkeiten, die in der SA-Matrix angelegt wurden, verifizieren zu können, müssen diese in die Petri-Netz-Strukturen überführt werden. Hierzu wird eine weitere Seite angelegt, in der die entsprechenden Abhängigkeiten als ServiceNet-Konstrukte angelegt werden. Auf alle Transitionen, Stellen und Seiten, die Teil einer Abhängigkeit sind, wird mithilfe von Referenzen in der Abhängigkeitsseite verwiesen. Im Folgenden werden nun die ServiceNet-Repräsentationen der in Abschnitt 4 erläuterten Relationen der SA-Matrix vorgestellt. Darauf folgend wird aufgezeigt, wie die Kombination der Relationen entsprechend der logischen Gruppen vorgenommen wird.

Die ServiceNet-Darstellung von Allow- Relationen ist in Abb. 9 gezeigt.[5] Eine Allow-Relation wird mithilfe einer Testkante (Arc mit ArcType read) modelliert, wie in Abb. 9 zu sehen (dargestellt mit je einer Pfeilspitze an jedem Ende einer Kante, hervorgehoben in rot). Damit wird gewährleistet, dass der Ziel-Zustandsübergang nur stattfinden kann, wenn der Quellzustand aktiv ist, ohne dass der Quellzustand dabei verlassen werden muss, entsprechend der Semantik der Allow-Relation.

Eine Prohibit-Abhängigkeit wird mittels einer Inhibitior-Kante vom Quell-Zustand in Form der entsprechenden Stelle (oben) zum Ziel-Zustandsübergang in Form der entsprechenden Transition (unten) modelliert, siehe Abb. 10. Somit kann der Ziel-Zustandsübergang nicht stattfinden, wenn der Quell-Zustand aktiv ist, was exakt dem Verhalten der Prohibit-Relation entspricht.

Mithilfe der Change-Relation soll ein Betriebsartenwechsel modelliert werden. Somit muss im entsprechenden ServiceNet ein Markenfluss von einem Zustand einer Betriebsart auf einen Zustand der anderen möglich gemacht werden. Dies wird mithilfe einer Transition zwi-

5 Der Übersicht halber werden die Abhängigkeiten hier nicht anhand der Referenzen gezeigt, sondern, wie sich die Wirkung im resultierenden Petri-Netz, also bei aufgelösten Referenzen, ergibt.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

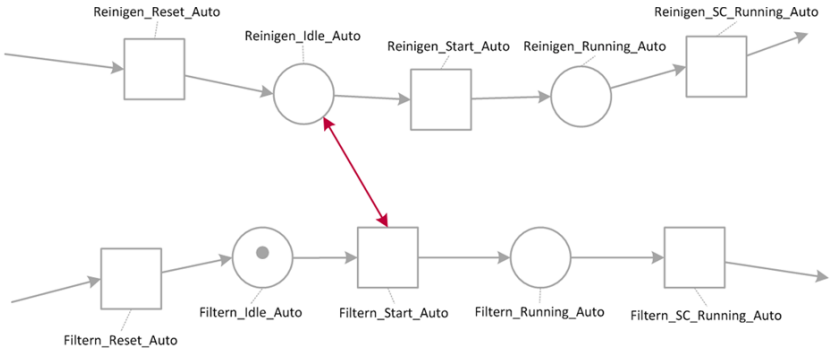


Abb. 9: Beispiel einer Allow-Relation.

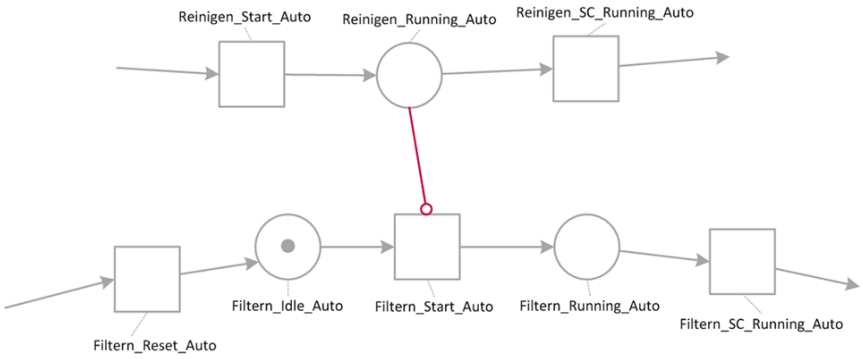


Abb. 10: Beipiel einer Prohibit-Relation.

schen den beiden Zuständen realisiert, wie beispielhaft in Abb. 11 dargestellt.

Die ServiceNet-Repräsentation einer Sync-Relation muss sicherstellen, dass die Ziel-Transition feuert, wenn im Quell-Netz der entsprechende Zustandsübergang stattfinden soll. Zudem muss dazu die Ziel-Transition auch aktiviert sein. Dieses Verhalten wird erreicht, indem je eine Kante von der Vorstelle der Quell-Transition auf die Ziel-Transition und von der Ziel-Transition zur Nachstelle der Quell-Transition angelegt wird. Dies ist in Abb. 12 dargestellt. Somit verlassen beide Teilnetze gleichzeitig den Präzustand und erreichen den Postzustand. Dies beinhaltet auch, dass der Zustandsübergang nicht stattfinden kann, wenn die Vorstelle des Ziel-Zustandsüberganges nicht mit einer Marke besetzt ist, bzw. der entsprechende Zustand nicht aktiv ist. Ist die Quell-Transition Ziel einer Reset-Kante, so besitzt sie keine Vorstelle, sondern eine Seite im Vorbereich. Hier ist genauso zu verfahren, mit dem Unterschied, dass die einzufügende Kante eine Reset-Kante ist und nicht von einem spezifischen Präzustand auf die Ziel-Transition gerichtet ist, sondern von der Seite auf die Ziel-Transition Somit genügt als Bedingung für den Zustandsübergang auch, dass ein beliebiger Zustand innerhalb der Seite aktiv ist. Um von der Seite, die die Abhängigkeiten enthält, auf diese Seite zu verweisen, ist in Abschnitt 5.2 die Referenz-Seite (RefPage) eingeführt worden.

Logische Verknüpfungen von Relationen müssen ebenfalls in ServiceNets dargestellt werden. Da logische Verknüpfungen von Change Relationen nicht benötigt werden, ist das Ziel einer UND-Verknüpfung stets ein Zustandsübergang bzw. eine Transition. Ein Ziel mit mehreren UND-verknüpften Relationen, also mehreren Relationen derselben logischen Gruppe in der SA-Matrix, wird daher stets als eine Transition dargestellt, auf die für jede UND-verknüpfte Relation je eine Kante gerichtet ist. Ein Ziel mit ODER-verknüpften Relationen, also Relationen unterschiedlicher logischer Gruppen in der SA-Matrix, wird hingegen als eine entsprechend häufig kopierte Transition auf die die Kanten gerichtet sind, repräsentiert. Folgendes Beispiel soll dies verdeutlichen:

Ein Zustandsübergang (*Filtern_Start_Auto*) soll nur stattfinden dürfen, wenn die Zustände *Abkoppeln_Idle_Auto* UND *Reinigen_Complete_Auto* aktiv sind ODER, wenn *Reinigen_Idle_Auto* aktiv ist. Diese Bedingung wird mithilfe von Allow-Relationen in der SA-Matrix modelliert. Die Relationen mit ersteren beiden Zuständen befinden sich somit in einer logischen Gruppe und letztere in einer anderen logischen Gruppe innerhalb der Spalte für *Fil-*

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

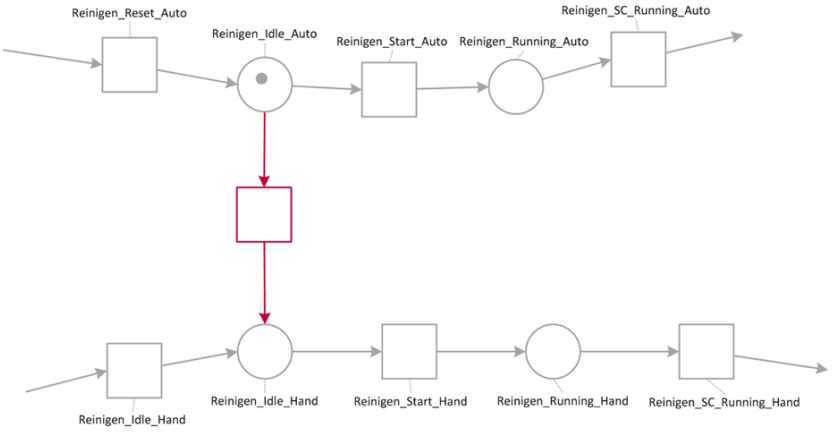


Abb. 11: Beispiel einer Change-Relation.

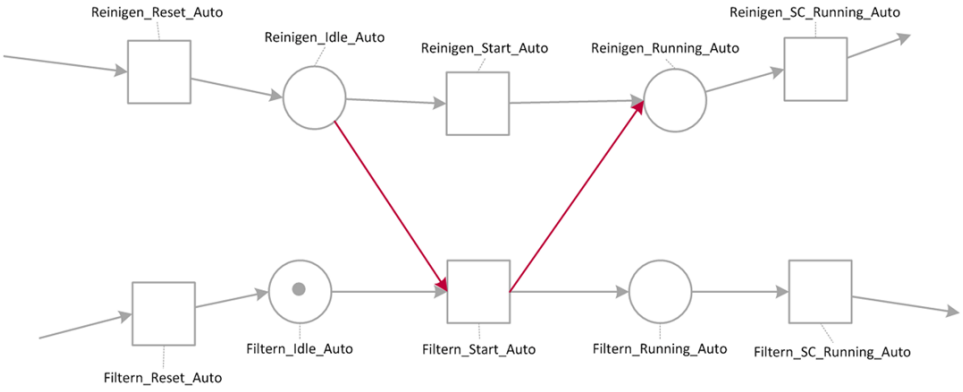


Abb. 12: Beispiel einer Sync-Relation.

tern_Start_Auto in der SA-Matrix. Die Repräsentation dieser Abhängigkeiten in einem ServiceNet ist dann wie in Abb. 13 dargestellt.

Sind die in der SA-Matrix entworfenen Abhängigkeiten in das Petri-Netz-Konstrukt überführt, so kann eine formale Analyse des gesamten Petri-Netzes stattfinden. Somit ist es z. B. möglich, das System auf Freiheit von Deadlocks zu überprüfen. Aber auch die Ermittlung der erreichbaren Zustände erlaubt dem Modulingenieur eine effiziente Überprüfung der Abhängigkeiten auf ihre Fehlerfreiheit. Zudem erlaubt die formale Modellierung das Anwenden weiterer formaler Methoden, wie bspw. das Model-Checking, zur Validierung spezieller und konkreter Anforderungen an die Serviceabhängigkeiten.

6 Fallstudie: Filtermodul

In diesem Abschnitt soll der in Abschnitt 3 vorgestellte Workflow anhand eines Fallbeispiels demonstriert werden. Hierbei handelt es sich um ein Filtermodul mit den Services *Filtern*, *Reinigen* und *Abkoppeln*. Alle diese Services können in den Betriebsarten *Hand* und *Automatik* betrieben werden. Dazu werden die Serviceabhängigkeiten dieses Moduls mit Hilfe der in Abschnitt 4 eingeführten Serviceabhängigkeitsmatrix modelliert. Für einzelne Abhängigkeiten wird außerdem die sich ergebende Petri-Netz-Struktur gezeigt, um den Einfluss auf das Verhalten der ServiceNets zu erläutern. Es kommt dabei eine prototypische Umsetzung der Methode, die aus einer Java-Applikation in Kombination mit einem Excel-Tool besteht, zum Einsatz. Dieser Prototyp erlaubt das Erstellen, Bearbeiten und Verifizieren der Abhängigkeiten.

6.1 Generieren der ServiceNets und Anlegen der Abhängigkeiten

Wie bereits oben erwähnt, müssen in einem ersten Schritt das Modul und seine Services definiert werden, damit die ServiceNets automatisch generiert werden können. Der

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

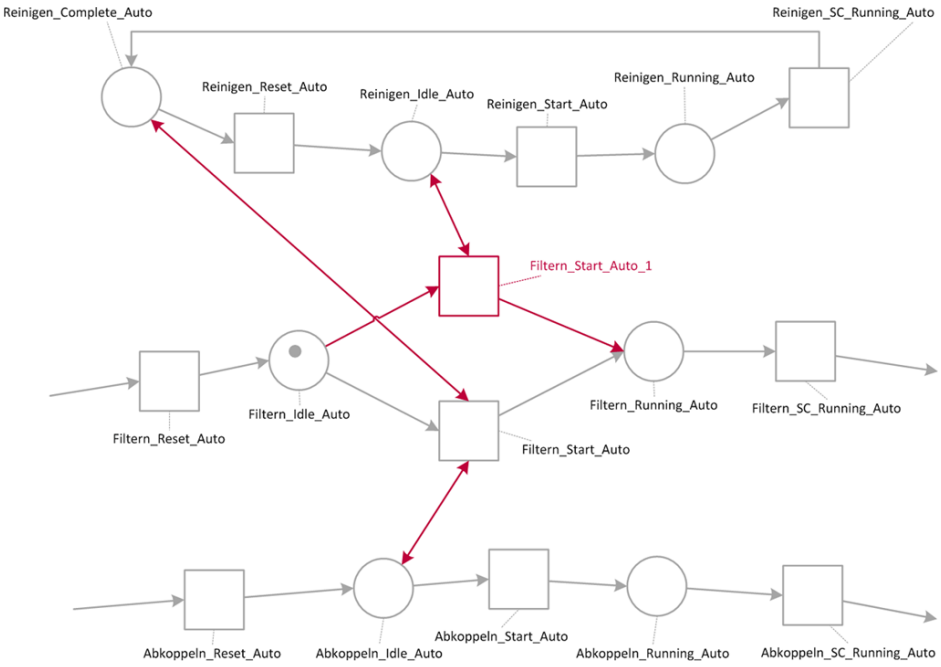


Abb. 13: Bespiel logischer Verknüpfungen von Relationen.

Modulingenieur gibt daher die Bezeichnung des Moduls sowie die Namen der Services und deren Betriebsarten in der entwickelten Applikation an. Die Applikation erzeugt daraus automatisch für jeden Service in jeder Betriebsart ein eigenes ServiceNet. Diese Netze befinden sich in einem gemeinsamen PNML-Dokument.

Zudem wird der Rumpf einer SA-Matrix in einer Excel-Datei erzeugt. Diese Datei enthält als Datenbasis die Bezeichnungen sämtlicher Stellen und Transitionen der zuvor erzeugten ServiceNets. Mit Hilfe dieser Daten kann der Anwender die zu erstellenden Abhängigkeiten bequem über Drop-Down-Listen anlegen. So wird die Eingabe fehlerhafter bzw. nicht vorhandener Abhängigkeitsquellen und -ziele vermieden. Für das betrachtete Filtermodul werden die folgenden Abhängigkeiten betrachtet:

1. Jeder Service soll im Automatikbetrieb nur gestartet werden können, wenn sich alle anderen Services ebenfalls in der Betriebsart Automatik und im Zustand Idle befinden.
2. Ein Betriebsartenwechsel soll jeweils nur vom Idle-Zustand der einen in den Idle-Zustand der anderen Betriebsart möglich sein.
3. Im Automatikbetrieb soll der Abbruchbefehl eines Services zu einem automatischen Abbruch aller Services führen.

Die erste Abhängigkeit entspricht einer Allow-Abhängigkeit. Für die Automatik-Startbefehle der Services müssen Allow-Relationen angelegt werden, welche die Initialzustände aller anderen Services als Quelle haben. Alle Relationen eines Startbefehles müssen einer logischen Gruppe angehören, um die UND-Verknüpfung zu realisieren.

Die zweite Abhängigkeit lässt sich vergleichsweise einfach mit Change-Relationen umsetzen. Diese müssen für jeden Service eine Verbindung von Idle des Automatikbetriebs zu Idle des Handbetriebs und umgekehrt erzeugen. Wie bereits in Abschnitt 4 erwähnt, existiert für Change-Relationen keine logische Verbindung, die Zuweisung einer Gruppe ist daher unerheblich.

Sync-Relationen kommen für die dritte Abhängigkeit zum Einsatz. Jeder Abort-Befehl muss mit den Abort-Befehlen aller anderen Services synchronisiert werden. In den Petri-Netzen werden dazu neue Kanten vom Vorbereich der Quelltransition zur Zieltransition und von der Zieltransition zum Nachbereich der Quelltransition angelegt. Da eine Abort-Transition auf eine Reset-Kante folgt und der Vorbereich somit eine ganze Seite darstellt, werden hierzu ReferencePages benötigt.

Abb. 14 zeigt die Abhängigkeitsmatrix, in der die drei genannten Abhängigkeiten erstellt wurden. Jede Spalte entspricht hier einem Abhängigkeitsziel und jedes Zeilenpaar einer Abhängigkeitsquelle. Die Zellen dazwischen enthalten die Relation (jeweils obere Zelle) sowie die logische Gruppe (untere Zelle) der Relation.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

Filtermodul

Abhängigkeits-Ziele →

Double click to verify		Filtern	Reinigen	Abkoppeln	Filtern	Reinigen	Abkoppeln	Filtern	Reinigen	Abkoppeln	Filtern	Reinigen	Abkoppeln
		Start_Auto	Start_Auto	Start_Auto	Idle_Hand	Idle_Hand	Idle_Hand	Idle_Auto	Idle_Auto	Idle_Auto	Abort_Auto	Abort_Auto	Abort_Auto
Abhängigkeits-Quellen	Filtern	Idle_Auto		Allow 1	Allow 1	Change 1							
	Reinigen	Idle_Auto	Allow 1		Allow 1		Change 1						
	Abkoppeln	Idle_Auto	Allow 1	Allow 1				Change 1					
	Filtern	Idle_Hand							Change 1				
	Reinigen	Idle_Hand								Change 1			
	Abkoppeln	Idle_Hand									Change 1		
	Filtern	Abort_Auto										Sync 1	Sync 1
	Reinigen	Abort_Auto									Sync 1		Sync 1
	Abkoppeln	Abort_Auto									Sync 1	Sync 1	

Abb. 14: Serviceabhängigkeitsmatrix zur Abbildung der oben genannten Abhängigkeiten in MS EXCEL.

Das verwendete Excel-Tool bietet die Möglichkeit, die angelegten Abhängigkeiten zu verifizieren. Dazu wird für jede Relation überprüft, ob diese die in Abschnitt 4 eingeführten Regeln bzgl. Quelle und Ziel der Relationstypen erfüllt. Auf diesem Weg wird sichergestellt, dass es durch falsche Modellierung nicht zu Fehlern bei der Erstellung der Petri-Netz-Konstrukte für die Abhängigkeiten kommt. So würde beispielsweise eine Change-Relation, deren Quelle und Ziel zur selben Betriebsart gehören, das ServiceNet unzulässig verändern. Durch die Verifikation lässt sich außerdem ausschließen, dass formale Fehler später die Analyse der Petri-Netze verfälschen.

Nachdem die SA-Matrix durch den Modulingenieur angelegt wurde, werden die Abhängigkeiten in PNML überführt. Dazu werden auf einer separaten Seite Petri-Netz-Konstrukte für die einzelnen Relationen angelegt. Innerhalb dieser Seiten werden Referenzstellen, -transitionen und -seiten verwendet, die auf die entsprechenden Objekte der ursprünglichen ServiceNets verweisen, vgl. Unterabschnitt 5.4. Um dies zu verdeutlichen, wird im Folgenden die konkrete Umsetzung einiger der in Abb. 14 modellierten Abhängigkeiten betrachtet.

Abb. 15 zeigt die drei Abhängigkeiten, welche die Stelle *Filtern_Idle_Auto* betreffen. Die auf *Filtern_Idle_Auto* verweisende Referenzstelle ist durch zwei Testkanten mit den Referenztransitionen *Reinigen_Start_Auto* sowie *Abkoppeln_Start_Auto* verbunden. Außerdem wurde eine neue Transition angelegt, die *Filtern_Idle_Auto* mit *Filtern_Idle_Hand* verbindet und somit einen Betriebsartenwechsel ermöglicht. Sie entspricht der Change-

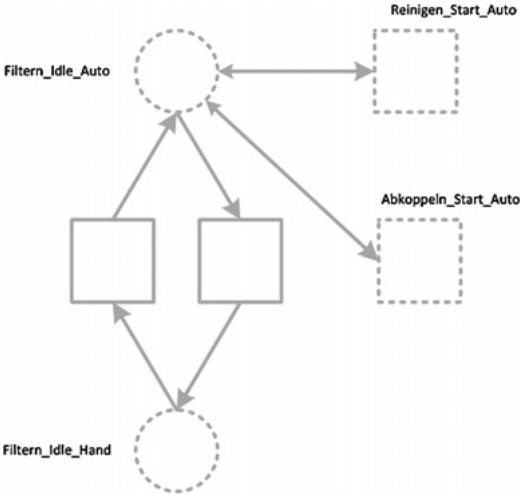


Abb. 15: Abhängigkeiten der Stelle *Filtern_Idle_Auto*.

Abhängigkeit in Zelle G6/7 der Tabelle in Abb. 14. Auch die entgegengesetzte Transition, die den Wechsel zurück in den Automatikbetrieb ermöglicht, wurde angelegt. Sie entspricht der Change-Relation in Zelle J12/13. *Filtern_Idle_Auto* ist das Ziel dieser Abhängigkeit.

Abb. 16 zeigt das Petri-Netz-Konstrukt des Abhängigkeitsziels *Filtern_Abort_Auto* (vgl. Spalte M in Abb. 14). Eine Referenzseite verweist auf die Seite *Filtern_StopPage_Auto*, den Vorbereich von *Filtern_Abort_Auto*. Von dieser Referenzseite führen Kanten zu Referenztransitionen, die auf die beiden Abhängigkeitsquellen verweisen. Diese Kanten sind Reset-Kanten, da ein Abbruch aus allen

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

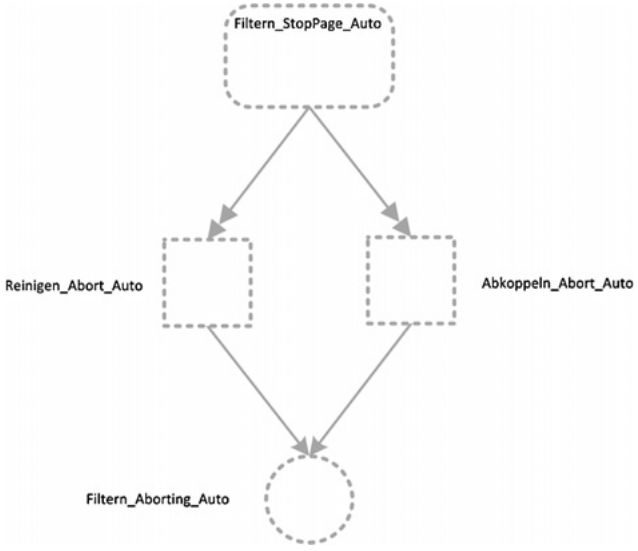


Abb. 16: Abhängigkeiten von *Filtern_Abort_Auto*.

Stellen innerhalb der Stop-Page möglich sein muss. Die beiden Quellen *Reinigen_Abort_Auto* und *Abkoppeln_Abort_Auto* sind somit jeweils mit zwei Seiten verbunden, da die bereits bestehenden Reset-Kanten aus dem ursprünglichen ServiceNet unberührt bleiben. Zwei weitere Kanten führen schließlich von den Abhängigkeitsquellen zu *Filtern_Aborting_Auto*, dem Nachbereich von *Filtern_Abort_Auto*. Diese Kanten stellen sicher, dass die Anzahl der Marken im Gesamtnetz konstant bleibt und auch durch das Auslösen von *Reinigen_Abort_Auto* oder *Abkoppeln_Abort_Auto* die Stelle *Filtern_Aborting_Auto* eine Markierung erhält.

Durch die angelegten Abhängigkeiten wurden sowohl die ServiceNets der einzelnen Betriebsarten eines Services als auch die unterschiedlichen Services miteinander verbunden. Es liegt nun ein zusammenhängendes Petri-Netz vor.

Für die im Anschluss folgende Analyse dieses Netzes müssen alle Referenzen aufgelöst werden, um die Analysemethoden für B/E-Netze anwenden zu können. Die an Referenzen gebundenen Kanten werden dazu an die jeweils referenzierten Objekte übertragen. Auch die Seiten und Reset-Kanten können nicht beibehalten werden. Um sie aufzulösen, muss jede Stelle innerhalb der Seite mit einer eigenen Transition verbunden werden, die eine Kopie der Reset-Transition darstellt. Falls wie in diesem Beispiel durch Abhängigkeiten mehrere Seiten zu einer Transition führen, so muss für jede Kombination aller Stellen aus den Seiten eine Kopie der Transition angelegt werden. Das Petri-Netz wird dadurch deutlich umfangreicher, ohne seine dynamischen Eigenschaften zu ändern.

6.2 Formale Analyse der Petri-Netze

Im letzten Schritt des präsentierten Workflows führt der Modulentwickler eine formale Analyse des resultierenden Petri-Netzes durch, um die Fehlerfreiheit der Abhängigkeiten und die Lauffähigkeit des Moduls sicherzustellen. Auch dies soll im Folgenden an einem Beispiel gezeigt werden. Dazu wird eine vereinfachte Form des obigen Beispiels betrachtet, in der das Filtermodul nur noch die beiden Services *Filtern* und *Reinigen* im Automatikbetrieb anbietet. Folgende Abhängigkeiten wurden bereits durch den Anwender erkannt und in der Serviceabhängigkeitsmatrix (siehe Abb. 17) modelliert:

Beide Services dürfen also nur starten, wenn der jeweils andere Service sich in den Zuständen *Idle*, *Aborted* oder *Stopped* befindet. Die drei Abhängigkeitsquellen der ersten beiden Abhängigkeitsziele befinden sich daher in unterschiedlichen logischen Gruppen. Wie im vorherigen Beispiel sind auch hier die beiden Abort-Befehle miteinander synchronisiert. Da das Modul nur noch eine Betriebsart anbietet, kann auf eine Change-Abhängigkeit verzichtet werden. Stattdessen wurde eine weitere Abhängigkeit durch den Modulingenieur festgelegt: Ein *Reset* aus *Aborted* soll nur möglich sein, wenn sich der andere Service in seinem sicheren Zustand, dem Initialzustand, befindet.

Für dieses Fallbeispiel wurde das aus diesen Abhängigkeiten resultierende Gesamtnetz mithilfe des frei zugänglichen Tools TINA[6] untersucht. Die Analyse erfordert eine Interpretation durch den Anwender. Sie kann mit dem gewünschten Verhalten verglichen werden und Hinweise auf etwaige Fehler geben.

In diesem konkreten Fall weist bereits eine Erreichbarkeitsanalyse den Modulingenieur auf ein schwerwiegendes Problem hin: Das Netz besitzt durch die eingefügten Abhängigkeiten einen Deadlock. Dieser Deadlock tritt auf, wenn beide Services den Zustand *Aborted* einnehmen. Dies wird bereits durch den Abbruchbefehl eines Services erreicht, da dieser aufgrund der Synchronisation auch den anderen Service zum Abbruch zwingt. Durch diesen synchronisierten Abbruch befinden sich beide Services zunächst in *Aborting* und anschließend in *Aborted*. Da ein *Reset* aus *Aborted* nur erlaubt wurde, wenn sich der jeweils andere Service im Initialzustand befindet, kann keiner der Services den Zustand *Aborted* verlassen.

Die beiden Abhängigkeiten der Abort-Synchronisation und Reset-Restriktion lassen sich nicht miteinander vereinbaren. Der Modulingenieur hat nun die Möglichkeit, die SA-Matrix zu überarbeiten. Eine Lösung ist es, die

6 TIme petri Net Analyzer, siehe http://projects.laas.fr/tina

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

		Filtermodul						
				Abhängigkeits-Ziele →				
		Double click to verify		Reinigen	Filtern	Reinigen	Filtern	Reinigen	Filtern
				Start_Auto	Start_Auto	Abort_Auto	Abort_Auto	Reset_Auto	Reset_Auto
Abhängigkeits-Quellen ↓		Filtern	Idle_Auto	Allow 1				Allow 1	
		Filtern	Aborted_Auto	Allow 2					
		Filtern	Stopped_Auto	Allow 3					
		Reinigen	Idle_Auto		Allow 1				Allow 1
		Reinigen	Aborted_Auto		Allow 2				
		Reinigen	Stopped_Auto		Allow 3				
		Filtern	Abort_Auto			Sync 1			
		Reinigen	Abort_Auto				Sync 1		

Abb. 17: Abhängigkeiten des vereinfachten Beispiels.

Reset-Bedingungen zu entfernen, damit der Zustand *Aborted* wieder verlassen werden kann. Eine erneute Überführung der so überarbeiteten SA-Matrix mithilfe der entwickelten Java-Applikation in das Petri-Netz erlaubt eine erneute Analyse. Diese Analyse bestätigt, dass das Netz nun reversibel ist und alle Zustände beliebig oft erreicht werden können.

7 Zusammenfassung und Ausblick

Dieser Beitrag hat eine Methode für das effiziente Engineering von Serviceabhängigkeiten während des Modulengineerings vorgestellt. Die Basis hierfür stellt die Serviceabhängigkeitsmatrix dar, in der die Abhängigkeiten mithilfe der vier Relationen Allow, Prohibit, Change und Sync entworfen werden können. Die Definition der SA-Matrix mithilfe eines Meta-Modells erlaubt dabei die Transformation der Abhängigkeiten in ein Petri-Netz-Konstrukt, welches die Zustandsautomaten eines jeden Services enthält. Zur effizienten Modellierung der Services und ihrer Abhängigkeiten wurde, basierend auf dem PNML-Kernmodell, ein neuer Petri-Netz-Typ definiert, die ServiceNets. Für jeden o. g. Abhängigkeitstyp sowie für logische Verknüpfungen der Abhängigkeiten wurden Darstellungen in ServiceNets aufgezeigt. Somit werden durch die Abhängigkeiten die Zustandsautomaten miteinander verknüpft, und es kann eine formale Analyse auf dem Petri-Netz zur Verifikation der Abhängigkeiten durchgeführt werden. Die Darstellung des Erreichbarkeitsgraphen kann zusätzlich den Modulingenieur bei der Verifikation unterstützen. Dank der Beschreibung in der standardisierten Beschreibungssprache PNML können diese Informationen automatisch von einem Zielsystem interpretiert werden. Die Machbarkeit der Methode wurde anhand einer Fallstudie aufgezeigt.

Zukünftige Arbeiten werden sich auf die Generierung von Quellcode zur Implementierung der Serviceabhängigkeiten in die Modulsteuerung konzentrieren. Auch werden zukünftig Abhängigkeiten zwischen Services verschiedener Module innerhalb einer Anlage untersucht werden müssen.

Literaturverzeichnis

1. CEFIC The European Chemical Industry Council: The european chemical industry – Facts and Figures. http://www.cefic.org/Facts-and-Figures/, 2014.
2. Bramsiepe, C.; Schembecker, G.: Die 50 %-Idee: Modularisierung im Planungsprozess. Chemie Ingenieur Technik, Vol. 84(5), S. 581–587, 2012.
3. M. Obst, T. Holm, S. Bleuel, U. Claussnitzer, L. Evertz, T. Jäger, T. Nekolla: *Automatisierung im Life Cycle modularer Anlagen: Welche Veränderungen und Chancen sich ergeben*. atp edition – Automatisierungstechnische Praxis (01–02), 2013, S. 24.
4. T. Holm: Aufwandsbewertung im Engineering modularer Prozessanlagen. Fortschritt-Berichte VDI-Reihe 20 Nr. 465: Rechnerunterstützte Verfahren. Düsseldorf: VDI Verlag 2016.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

5. Forschungsunion/acatech: Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0 – Abschlussbericht des Arbeitskreises Industrie 4.0., 2013.
6. DECHEMA e.V.: Modular Plants: Flexible chemical production by modularization and standardization – status quo and future trends, 2016.
7. J. Bernshausen, A. Haller, T. Holm, M. Hoernicke, M. Obst, J. Ladiges: *Namur Modul Type Package – Definition: Beschreibungsmittel für die Automation modularer Anlagen*. atp edition – Automatisierungstechnische Praxis, Vol. 58(1–2), 2016, S. 72–81.
8. M. Hoernicke, T. Holm, A. Haller, J. Bernshausen, D. Schulz, T. Albers, C. Kotsch, M. Maurmaier, A. Stutz, H. Bloch, S. Hensel: Technologiebewertung zur Beschreibung für verfahrenstechnische Module – Ergebnisse des Namur AK 1.12.1. In: Kongress Automation 2016, Baden-Baden, 07.-08. Juni 2016.
9. T. Holm, M. Obst, J. Ladiges, L. Urbas, A. Fay, T. Albers, U. Hempen: *Namur Modul Type Package – Implementierung: Anwendung des Namur-MTP für Prozessanlagen*. atp edition – Automatisierungstechnische Praxis, Vol. 58(1–2), 2016, S. 72–81.
10. H. Bloch, S. Hensel, M. Hoernicke, K. Stark, A. Menschner, L. Urbas, A. Fay, T. Knohl, J. Bernshausen, A. Haller: *Zustandsbasierte Führung modularer Prozessanlagen*. In: atp edition Ausgabe 10/2017, Seite 46–57.
11. H. Bloch, A. Fay, M. Hoernicke: *Analysis of service-oriented architecture approaches suitable for modular process automation*. IEEE International Conference on Emerging Technology & Factory Automation (ETFA 2016), Berlin, September 6–9, 2016.
12. H. Bloch, M. Hoernicke, S. Hensel, A. Hahn, A. Fay, L. Urbas, T. Knohl, J. Bernshausen: *A Microservice-Based Architecture Approach for the Automation of Modular Process Plants*. In: ETFA 22nd IEEE International Conference on Emerging Technologies And Factory Automation, September 12–15, 2017, Limassol, Cyprus.
13. DIN EN 61512-1. Chargenorientierte Fahrweise – Teil 1: Modelle und Terminologie, 2000.
14. H. Bloch, A. Fay, S. Hensel, A. Hahn, L. Urbas, S. Wassilew, M. Hoernicke, T. Knohl, J. Bernshausen, A. Haller: *Model-based Engineering of CPPS in the process industries*. In: IEEE 15th International Conference of Industrial Informatics INDIN'2017, July 24–26, 2017, Emden, Germany.
15. ANSI/ISA-TR22.00.02-2015. Machine and Unit States: An implementation example of ANSI/ISA-88.00.01, 2015.
16. T. Holm, J. Ladiges, S. Wassilew, P. Altmann, A. Fay, L. Urbas, U. Hempen: *DIMA im realen Einsatz: Von der Idee zum Prototypen*. In: Kongress Automation, Baden-Baden, 07.-08. Juni 2016.
17. IEC 62881 Ed. 1.0 (draft). Cause & Effect Table, 2016.
18. ISO/IEC 15909-2. Systems and software engineering – High-level Petri nets – Part2: Transfer format, 2011.
19. W. Reisig, K. Schmidt, C. Stahl, *Kommunizierende Workflow-Services modellieren und analysieren*. Informatik – Forschung und Entwicklung, Vol. 20(1), 2005, S. 90–101.
20. S. Buchwald, T. Bauer, *Modellierung von Service-Aufrufbeziehungen zwischen prozessorientierten Applikationen*. EMISA Forum, Vol. 30(2), 2010, S. 32–48.
21. M. Becker, S. Klingner, *Formalisierung von Regeln zur Darstellung von Abhängigkeiten zwischen Elementen von Product-Service-Systems*. Universität Leipzig, 2012.
22. B. Heinrich; M. Klier, S. Zimmermann, *Automatisierte Modellierung, Umsetzung und Ausführung von Prozessen – Ein Web Service-basiertes Konzept*, Wirtschaftsinformatik Proceedings 2011.
23. C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz, *Reference Model for Service Oriented Architecture 1.0*., Stand vom 12. Oktober 2006. Verfügbar: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html (5. Februar 2018).
24. H. Cervantes, R. S. Hall, *Automating service dependency management in a service-oriented component model*. ICSE CBSE Workshop. 2003.
25. B. Li, *Managing Dependencies in Component-Based Systems Based on Matrix Model*. In: Proc. Of Net. Object. Days 2003, S. 22–25, 2003.
26. V. Dubinin, V. Vyatkin, T. Pfeiffer, *Engineering of Validatable Automation Systems Based on an Extension of UML Combined With Function Blocks of IEC 61499*, Proceedings of the IEEE International Conference on Robotics and Automation, 2005, pp. 3996–4001.
27. C. Reichmann, P. Graf, K. D. Müller-Glaser, *Automatisierte Modellkopplung heterogener eingebetteter Systeme*. In: Holleczek P., Vogel-Heuser B. (Eds) *Eingebettete Systeme. Informatik aktuell*. Springer, Berlin, Heidelberg, 2004.
28. O. Fengler, W. Fengler, V. Duridanova, *Modeling of complex automation systems using colored State Charts*, Proceedings of the IEEE International Conference on Robotics and Automation, 2002.
29. T. J. Prati, J. M. Farines, M. H. de Queiroz, *Automatic test of safety specifications for PLC programs in the Oil and Gas Industry*, IFAC-PapersOnLine, Vol. 48(bissue6), 2015, pp. 27–32.
30. T. Stahl, M. Völter, J. Bettin: *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. 1. Aufl. Heidelberg: dpunkt-Verl., 2005.
31. IEC 61131-3: *Programmable controllers – Part 3: Programming languages*, 2013.
32. R. Drath, A. Fay, T.Schmidberger: *Computer-aided design and implementation of interlocking control code*. In: 2006 IEEE International Symposium on Computer-Aided Control Systems Design (CACSD'06), München, 04.–06. 10. 2006, ISBN: 0-7803-9797-5, pp. 2653–2658.
33. R. David, H. Alla: *Discrete, Continous, and Hybrid Petri Nets*. Springer-Verlag, Berlin, Heidelberg, 2005.

Autoreninformationen

M. Sc. Jan Ladiges
Helmut-Schmidt-Universität, Hamburg, Germany
jan.ladiges@hsu-hh.de

M. Sc. Jan Ladiges war wissenschaftlicher Mitarbeiter am Institut für Automatisierungstechnik der Helmut-Schmidt-Universität Hamburg. Seine Forschungsschwerpunkte waren die automatisierte Bewertung evolvierender Fertigungssysteme mithilfe aus Daten generierter Petri-Netze sowie die modulare Prozessautomation.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43

M. Sc. Aljosha Köcher
Helmut-Schmidt-Universität, Hamburg, Germany
aljosha.koecher@hsu-hh.de

M. Sc. Aljosha Köcher forschte im Rahmen seines Wirtschaftsingenieur-Studiums am Institut für Automatisierungstechnik der Helmut-Schmidt-Universität Hamburg.

B. Sc. Peer Clement
Helmut-Schmidt-Universität, Hamburg, Germany
peer.clement@hsu-hh.de

B. Sc. Peer Clement forscht im Rahmen seines Wirtschaftsingenieur-Studiums am Institut für Automatisierungstechnik der Helmut-Schmidt-Universität Hamburg.

M. Sc. Henry Bloch
Helmut-Schmidt-Universität, Hamburg, Germany
henry.bloch@hsu-hh.de

M. Sc. Henry Bloch arbeitet seit Juli 2015 als wissenschaftlicher Mitarbeiter am Institut für Automatisierungstechnik der Helmut-Schmidt-Universität Hamburg bei Herrn Prof. Dr.-Ing. Alexander Fay. Sein Forschungsschwerpunkt liegt in der Entwicklung von Steuerungskonzepten modularer Prozessanlagen.

Dr.-Ing. Thomas Holm
WAGO Kontakttechnik GmbH & Co.KG, Minden, Germany
Thomas.Holm@wago.com

Dr-Ing Thomas Holm ist Head of Innovation & Technology bei WAGO Kontakttechnik GmbH & Co.KG, Minden.

Prof. Dr.-Ing. Paul Altmann
Technische Universität Dresden, Dresden, Germany
paul.altmann@tu-dresden.de

Prof. Dr.-Ing. Paul Altmann ist wissenschaftlicher Mitarbeiter an der Professur für Prozessleittechnik an der Technischen Universität Dresden.

Prof. Dr.-Ing. Alexander Fay
Helmut-Schmidt-Universität, Hamburg, Germany
alexander.fay@hsu-hh.de

Prof. Dr.-Ing. Alexander Fay ist Professor für Automatisierungstechnik an der Fakultät für Maschinenbau der Helmut-Schmidt- Universität/Universität der Bundeswehr Hamburg. Sein Forschungsschwerpunkt sind Beschreibungsmittel, Methoden und Werkzeuge für ein effizientes Engineering von Automatisierungssystemen.

Prof. Dr.-Ing. Leon Urbas
Technische Universität Dresden, Dresden, Germany
leon.urbas@tu-dresden.de

Prof. Dr.-Ing. Leon Urbas (geb. 1965) ist Inhaber der Professur für Prozessleittechnik an der Technischen Universität Dresden. Seine Hauptarbeitsgebiete beim Engineering verteilter sicherheitskritischer Systeme sind Funktionsintegration, modellgetriebenes Engineering, Modularisierung, Informationsmodelle der Prozessindustrie und Middleware in der Automatisierungstechnik.

Bereitgestellt von | Saechsische Landesbibliothek - Staats- und Universitaetsbibliothek Dresden (SLUB)
Angemeldet
Heruntergeladen am | 21.11.18 09:43