Formale Analyse des Zeitverhaltens Netzbasierter Automatisierungssysteme

Dipl.-Ing. Jürgen Andreas Greifeneder
geb. in Schwäbisch Hall

Vom Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

Eingereicht am:	19. September 2007
Tag der mündlichen Prüfung:	02. November 2007
Dekan des Fachbereichs:	Prof. Dr.-Ing. Steven Liu

Promotionskommission	
Vorsitzender:	Prof. Dr. techn. Gerhard Fohler
Berichterstattende:	Jun. Prof. Dr.-Ing. Georg Frey
	Prof. Dr.-Ing. Birgit Vogel-Heuser

D 386

Kurzfassung

Die Architekturen vieler technischer Systeme sind derzeit im Umbruch. Der fortschreitende Einsatz von Netzwerken aus intelligenten rechnenden Knoten führt zu neuen Anforderungen an den Entwurf und die Analyse der resultierenden Systeme. Dabei spielt die Analyse des Zeitverhaltens mit seinen Bezügen zu Sicherheit und Performanz eine zentrale Rolle. Netzbasierte Automatisierungssysteme (NAS) unterscheiden sich hierbei von anderen verteilten Echtzeitsystemen durch ihr zyklisches Komponentenverhalten. Das aus der asynchronen Verknüpfung entstehende Gesamtverhalten ist mit klassischen Methoden kaum analysierbar. Zur Analyse von NAS wird deshalb der Einsatz der wahrscheinlichkeitsbasierten Modellverifikation (PMC) vorgeschlagen. PMC erlaubt detaillierte, quantitative Aussagen über das Systemverhalten. Für die dazu notwendige Modellierung des Systems auf Basis wahrscheinlichkeitsbasierter, zeitbewerteter Automaten wird die Beschreibungssprache DesLaNAS eingeführt. Exemplarisch werden der Einfluss verschiedener Komponenten und Verhaltensmodi auf die Antwortzeit eines NAS untersucht und die Ergebnisse mittels Labormessungen validiert.

Currently the architectures of many technical systems are undergoing considerable changes. The increasing use of networks connecting intelligent processor nodes leads to new requirements on the design and the analysis of the resulting systems. In this process, the analysis of temporal behavior with regards to safety and performance plays a central role. Networked Automation Systems (NAS) differ from other distributed real-time systems due to the cyclic behavior of their components. The overall behavior arising from the asynchronous composition of these components is hardly analyzable with traditional methods. Therefore, the use of Probabilistic Model Checking (PMC) is proposed for the analysis of NAS. PMC allows detailed quantitative statements about the overall system behavior. For the modeling task, which is based on the use of probabilistic timed automata, the description language DesLaNAS is introduced. As a case study, the influence of different components and behavior modes on the response time of a typical NAS is analyzed. The results are validated by measured values.

Meinen Eltern Helmut und Ingrid Greifeneder
in Würdigung der mir durch sie erschaffenen
Chance, dieses Bildungsniveau zu erreichen.

Wenn Du ein Schiff bauen willst, so trommle nicht Männer zusammen, um Holz zu beschaffen, Werkzeuge vorzubereiten, Aufgaben zu vergeben und die Arbeit einzuteilen, sondern lehre sie die Sehnsucht nach dem weiten endlosen Meer.

Antoine de Saint-Exupéry (zugeschrieben, Quelle unklar)

Danksagung

Wenn der Wanderer den Gipfel erreicht, erwartet ihn ein Gefühl der Freiheit und mit etwas Glück eine grandiose Aussicht. Zeit für ein Berg heil, Zeit, tief durchzuatmen, Zeit, zu danken. Jeder war in seiner Art wichtig und verdiente es, erwähnt zu sein. Doch möchte ich mich, stellvertretend für alle ungenannten, auf wenige beschränken.

Es waren François und Ursula Cellier, die mich zur Promotion ermutigten und stets nach meinen Fortschritten fragten. Es waren meine Eltern und Geschwister, auf deren Unterstützung ich jederzeit uneingeschränkt zählen konnte. Und es waren meine Freunde, deren wichtigste Aufgabe darin bestand, da zu sein. Auch waren da jene, die mir auf Konferenzen zuhörten und Anregung und Motivation gaben. Und Elke, Florian, Helmut, Kerstin, Matthias, Stephanie und Thomas, die mir beim Korrigieren halfen.

Diese Dissertation schreiben zu dürfen war eine Herausforderung, aber auch ein Privileg, für dessen Finanzierung ich der Stiftung der Deutschen Wirtschaft (sdw), meinen Eltern und der Technischen Universität Kaiserslautern danken möchte.

Danken möchte ich auch jenen, mit denen ich im Rahmen verschiedenster Aktivitäten einen Ausgleich zur Forschungsarbeit finden durfte. Danken möchte ich den Entwicklern der Software PRISM, die sich viel Zeit für Diskussionen nahmen, sowie den Forscherteams des LURPA der ENS Cachan (F) und des Departamento de Engenharia Elétrica der Universidade de Brasília (BR) für deren Gastfreundschaft. Danken möchte ich aber auch der kompletten Kaiserslauterer Arbeitsgruppe, in deren Mitte nicht nur die Arbeit Spaß machte, sondern ich auch viele schöne Ausflüge und andere Aktivitäten erleben durfte.

Frau Prof. Dr.-Ing. Birgit Vogel-Heuser und Herrn Jun. Prof. Dr.-Ing. Georg Frey möchte ich für das Interesse an meiner Arbeit und all die Zeit und Mühe, die sie in die Begutachtung investiert haben, danken. Gleiches gilt für den Vorsitzenden der Promotionskommission, Herrn Prof. Dr. techn. Gerhard Fohler.

Schlussendlich gilt mein Dank meinem Doktorvater Georg Frey für die wissenschaftliche und menschliche Begleitung meines akademischen Weges.

Inhaltsverzeichnis

1 Einleitung

> *„What is different today is that technology can put low-cost processing power at remote locations via microprocessors and that information can be transmitted reliably via shared digital networks or even wireless connections.“*

Was [Antsaklis und Baillieul, 2007] hier in nüchternen Worten beschreiben, ist nicht weniger als eine der größten technischen Veränderungen unserer Zeit. Traditionell war die Automatisierungstechnik durch eine enge, sowohl funktionale als auch räumliche Kopplung von Prozess und Regelung bzw. Steuerung gekennzeichnet. Daran änderte sich auch wenig, als man in der Lage war, das System in gekoppelte Teilsysteme zu zerlegen und für jedes dieser Teilsysteme eigenständige Regler zu implementieren. Erst mit der Einführung von netzbasierten, dezentralen Kommunikationsstrukturen wurde die Möglichkeit zur Trennung geschaffen. Entstanden sind dabei die als Netzbasierte Automatisierungssysteme (Networked Automation Systems, NAS) bezeichneten Architekturen. Derartige Systeme bestehen aus einem oder mehreren Controllern (i.A. Speicherprogrammierbaren Steuerungen, SPS), einer meist größeren Anzahl an Sensoren und Stellgliedern sowie einem diese einzelnen Komponenten miteinander verbindenden Netzwerk inklusive der zugehörigen Netzwerkschnittstellenhardware (I/O-Karten). Das Besondere hieran ist, dass nicht nur das Netz von einer Anzahl unabhängiger Nutzer verwendet wird, sondern jede der SPSen von jedem Sensor Signale empfangen, an jede andere SPS Informationen senden und – zumindest theoretisch – an jedes Stellglied Anweisungen übermitteln kann.

Diese Anordnung bringt eine ganze Reihe von Vorteilen mit sich, wie z.B. eine bessere, da gemeinsame Nutzung vorhandener Ressourcen (z.B. Sensoren, Verkabelung), eine gesteigerte Effizienz und Überwachbarkeit der Anlage durch horizontal wie vertikal integrierte komplexe, dezentrale Kontrollstrukturen, der daraus folgenden größeren Flexibilität und schließlich der Möglichkeit zur örtlichen Trennung von Abläufen, Reglern untereinander als auch von Reglern und Prozess. Verwendet man für die Netzwerkstruktur Ethernet, kommen noch stetig sinkende Preise für die Hardware, eine nachhaltige Verbesserung von Qualität und Umfang der angebotenen Technologie und die Möglichkeit, Kommunikationsschranken zu überwinden, hinzu. Betriebswirtschaftlich gesehen sprechen die deutlich verminderten Verkabelungs- und Neukonfigurationskosten, einfache Austauschbarkeit der Komponenten und Erweiterbarkeit der Anlage sowie die Möglichkeit, Komponenten unterschiedlicher Hersteller zu kombinieren, eine eindeutige Sprache. Dies gilt um so mehr, da in herkömmlichen Strukturen zusätzliche Komponenten nicht oder nur schwer integrierbar sind, wenn sich die Anforderungen geändert haben.

Der Preis für all diese Vorzüge ist, dass die entstehenden Strukturen um ein Vielfaches komplexer sind als bisherige Architekturen. Diese Komplexität rührt dabei nicht nur von der Vielzahl unterschiedlicher zu einem Gesamtsystem vereinigten Komponenten her, sondern resultiert gerade aus der Tatsache, dass diese unabhängigen Komponenten zugleich auch ein Ganzes bilden. Jede Komponente bringt hierbei ihre eigenen Verhaltenseigenschaften

in das System mit ein, wodurch sich das dynamische Verhalten des Gesamtsystems aus dem komplexen Zusammenspiel aller Einzeleigenschaften ergibt.

In der Folge bedeutet dies, dass es für die Analyse derartiger Systeme nicht mehr ausreicht, die reinen automatisierungstechnischen Aspekte zu konsolidieren, sondern das Zusammenspiel aus Berechnungsabläufen (z.B. Signalaufbereitung, Steuerungsalgorithmus), Kommunikation (z.B. Netzübertragung) und Regelungstechnik berücksichtigt werden muss. Im Fachjargon wird dieses Konglomerat als C^3-Technologie (computation, communication, control) bezeichnet und ist derzeit eine der wichtigsten Herausforderungen der Automatisierungstechnik. So listet beispielsweise das amerikanische NSF-Panel on Future Directions in Control, Dynamics and Systems diese Problematik auf Platz eins ihrer Empfehlungen zukünftiger Forschungsförderprogramme [Murray et al., 2003], und die europäische Kommission kommt in ihrem Bericht zum Workshop on future and emerging control systems [IST-Programme, 2000] gar zu der Aussage:

> *„Questions about the convergence of C3 technologies [...] lead today to similar questions about the convergence of disciplines“*

Motivation

Aus dem oben Beschriebenen ergibt sich die zwingende Notwendigkeit, dieses Thema ernst zu nehmen und die bisher noch nicht zufriedenstellend gelöste Frage nach Verlässlichkeit und Performanz derartiger Systeme anzugehen. Ähnlich der von [Kopetz, 2003] vorgestellten Korrektheitsklassen lassen sich die mit Verlässlichkeit und Performanz in Beziehung stehenden Fragestellungen auf wertbasierte und zeitbasierte Eigenschaften aufteilen: reagiert das System korrekt (wertbasiert) und reagiert das System innerhalb der zulässigen temporalen Grenzen (zeitbasiert). Während für die Untersuchung von wertbasierten Eigenschaften schon umfangreiche Analysemethoden zur Verfügung stehen, ist der temporale Bereich in der Automatisierungstechnik bisher wenig erforscht. Ein wichtiges Charakteristikum temporaler Fragestellungen ist es, dass nicht das System selbst analysiert wird, sondern die auf einem solchen System laufenden Prozesse Gegenstand der Analyse sind. Als Voraussetzung für die Analyse von Prozessen muss jedoch der Prozessablauf auf dem System mitverfolgt werden (Signal-Tracking). Der Zeit kommt hierbei doppelte Bedeutung zu: Auf der einen Seite ist sie als das Systemverhalten beschreibende Modellvariable unerlässlich, auf der anderen Seite stellt sie neben der reinen Funktionsanalyse die wichtigste Säule der temporalen Analyse dar.

Zur Analyse temporaler Eigenschaften ist es daher einerseits notwendig, Modelle zu verwenden, die mit den unterschiedlichen Aspekten eines NAS umgehen können, wie z.B. Zeitverzögerungen, Nebenläufigkeit, deterministische Abhängigkeiten oder Stochastik. Andererseits bedarf es einer Methodik, die Antworten auf jene Fragen liefern kann, welche die temporalen Eigenschaften von NAS betreffen. Charakteristisch daran ist, dass sich all diese Fragestellungen auf eine Kopplung von zeitlichen (reagiert das System innerhalb vorgegebener Zeitschranken) und stochastischen (mit welcher Wahrscheinlichkeit geschieht

dies) Aspekten zurückführen lassen. Stochastische Effekte treten in NAS in vier verschiedenen Formen auf. Erstens erfordert in NAS auftretendes unvorhersagbares Verhalten, wie z.B. Paketverluste und Ausfälle, die Einbeziehung stochastischer Beschreibungsformen in die Modellierung. Da selbst mit geringer Wahrscheinlichkeit auftretende Effekte große Verzögerungen nach sich ziehen können, dürfen sie in der Analyse keinesfalls vernachlässigt werden. Zweitens gibt es in einem NAS Komponenten, wie z.B. das Netzwerk, deren Abläufe zwar im Detail bekannt sind und daher problemlos modellierbar wären, in ihrer eigentlichen Funktion jedoch kaum vom betrachteten Prozess abhängig sind. Konsequenterweise lässt sich das Verhalten solcher Komponenten am besten durch stochastische Zeitverteilungen abstrahiert ins System einbringen. Drittens finden stochastische Beschreibungsformen auch dort Anwendung, wo Konflikte auftreten. Dies ist z.B. der Fall, wenn zwei Komponenten nahezu gleichzeitig einen Dienst einer dritten Komponente anfordern. Und viertens kommt man auch dann nicht ohne stochastische Anteile aus, wenn keiner der vorgenannten Aspekte in der Modellierung berücksichtigt werden muss. Dies liegt daran, dass der Beginn einer Prozessbeobachtung selten mit dem Systemstart zusammenfällt, das System also schon am Laufen ist, wenn der zur Analyse anstehende Prozess beginnt. Da einerseits die einzelnen Komponenten eines NAS i.d.R. nicht miteinander synchronisiert sind, und andererseits Abläufe meist durch Sensorwerte physikalischer Prozesse (wie z.B. Bewegung, Temperatur) ausgelöst werden, ergibt sich der Systemzustand zum Auslösezeitpunkt als stochastische Größe, die bei der Analyse berücksichtigt werden muss. Aus diesem Grunde ist es – selbst für Systeme, in denen die drei erstgenannten Effekte nicht auftreten – unerlässlich, eine Modellierungsmethodik zu verwenden, die der Notwendigkeit Rechnung trägt, stochastische Elemente zu integrieren.

Auch wenn heute bereits die ersten Anlagen auf NAS-Basis arbeiten und sich Ethernet als komplexe Kommunikationsbasis immer größerer Beliebtheit erfreut, darf man sich nicht darüber hinwegtäuschen lassen, dass die großen Umwälzungen noch bevorstehen. So steht z.B. die Energiewirtschaft vor der Einführung der busbasierten Stationsleittechnik nach [IEC 61850]. Das Besondere daran ist, dass hier Anwendungsgebiete dauerhaft gekoppelt werden, deren zugrunde liegende Technologien unterschiedlich schnell weiterentwickelt werden [Brand, 2003]. Die hieraus folgenden Anforderungen an die Automatisierungstechnik machen nach [Bretthauer, 2005] die Erarbeitung von neuen Methoden zum systematischen Entwurf von Systemen mit gewünschten Eigenschaften erforderlich. Schlussendlich sollte nicht verschwiegen werden, dass sich mit den technologischen Möglichkeiten auch die Produktionsabläufe und Rückkopplungen auf den Menschen ändern werden, wie beispielsweise in [Greifeneder et al., 2003] für eine kundenorientierte Fahrradproduktion beschrieben.

Lösungsansätze und Aufbau der Arbeit

Ziel dieser Arbeit ist es, eine Methodik zu präsentieren, mittels derer man Vor- und Nachteile verschiedener Systemarchitekturen quantitativ bewerten, Qualitätsmerkmale analytisch optimieren, System-Konfigurationen offline evaluieren sowie die Folgen des gerade in Ethernetstrukturen einfachen Hinzuklemmens weiterer Komponenten abschätzen kann.

Zunächst führt Kapitel 2 die Netzbasierten Automatisierungssysteme ein und stellt dar, wie durch die Analyse von Antwortzeiten Verfügbarkeits- und Performanzeigenschaften eines Systems (bzw. der darauf laufenden Prozesse) quantifiziert werden können. Die Bewertung literaturbekannter Ansätze zur Antwortzeitanalyse zeigt, dass für die Untersuchung von Netzbasierten Automatisierungssystemen die Verwendung der wahrscheinlichkeitsbasierten Modellverifikation (PMC) optimal ist. PMC wird daher in Kapitel 3 detailliert vorgestellt und in Kapitel 4 auf die bereits angesprochene Problematik erweitert, den Systemzustand eines NAS zum Zeitpunkt des Eintrittes eines externen Ereignisses stochastisch zu bestimmen.

PMC ist eine aus dem Gebiet der formalen Analyse stammende Methodik. Daher sind die zugehörigen Programme für den anwendungsorientierten Ingenieur eher schwer zu implementieren. Deutlich besser geeignet wäre ein auf generischen Modulen aufbauender Modellierungsprozess, der sich auf Instanziierung und Parametrierung notwendiger Modulblöcke sowie des für die Prozessbeobachtung notwendigen Signal-Trackings beschränkt. Dieser Vision folgend erläutert Kapitel 5 zunächst die unterschiedlichen in einem NAS auftretenden Verhaltensformen und leitet hieraus eine dem obigen Modellierungsanspruch nachkommende Beschreibungssprache ab: DesLaNAS (Description Language for Networked Automation Systems). Die in DesLaNAS erstellten Modelle werden dann erstens auf eine eigens hierfür definierte, zeitkontinuierliche Automatenstruktur abgebildet, zweitens zur Genügung der durch Rechenleistung und Hauptspeicher gegebenen Einschränkungen diskretisiert und drittens in den Quellcode der für die Analyse verwendeten Software PRISM [Kwiatkowska et al., 2002] umgesetzt. Selbstverständlich kann die Beschreibungssprache DesLaNAS nicht nur genutzt werden, um PRISM-Modelle zu erstellen, sondern sie bildet insbesondere eine von der konkreten Softwareplattform und Analysemethodik unabhängige Beschreibungsform Netzbasierter Automatisierungssysteme.

Zur Veranschaulichung führt Kapitel 6 ein typisches Anwendungsbeispiel ein und diskutiert hieran verschiedene Aspekte eines NAS. Dabei wird untersucht, welchen relativen Anteil das Netzwerk sowie die durch Überlagerung von Ausführungszyklen zustande kommenden Synchronisationseffekte an der mittleren Antwortzeit haben. Die Herleitung und Diskussion der hierzu notwendigen DesLaNAS-Modelle wird ebenso dargestellt wie ein Vergleich der analytisch erzielten Antwortzeitverteilungsgraphen mit messtechnischen Ergebnissen. Darüber hinaus wird vorgestellt, wie man aus Kenntnis der Antwortzeitverteilung Konfigurationsentscheidungen ableiten kann und welchen Einfluss Komponentenfehler und Zugriffskonflikte auf die Antwortzeit – und somit auf die Performanz – eines NAS haben. Kapitel 7 gibt schließlich eine kurze Zusammenfassung und einen Ausblick auf zukünftige Forschungsarbeiten.

2 Netzbasierte Automatisierungssysteme (NAS)

In der Industrie werden Prozesse größtenteils unter Verwendung Speicherprogrammierbarer Steuerungen (SPS) automatisiert. Während dies traditionell fest verdrahtete Systeme waren, führen steigende Ansprüche an die Flexibilität und Modularität von Automatisierungssystemen zu einer Hinwendung zu neuen dezentralen, vernetzten Strukturen. Die schnelle Ausbreitung von Ethernet-Technologien und der damit einhergehende Verfall der Preise der zugehörigen Komponenten versprechen eine rapide Steigerung der Nutzung dieser offenen Technologie durch den Automatisierungssektor. Ausgelöst durch die Verschmelzung von Netzwerktechnologien und klassischer Automatisierungstechnik entstehen Netzbasierte Automatisierungssysteme (Networked Automation Systems, NAS). Diese Systeme sind insbesondere dadurch gekennzeichnet, dass sie die, den verschiedenen Komponenten innewohnenden, Antwortzeitverhalten miteinander koppeln.

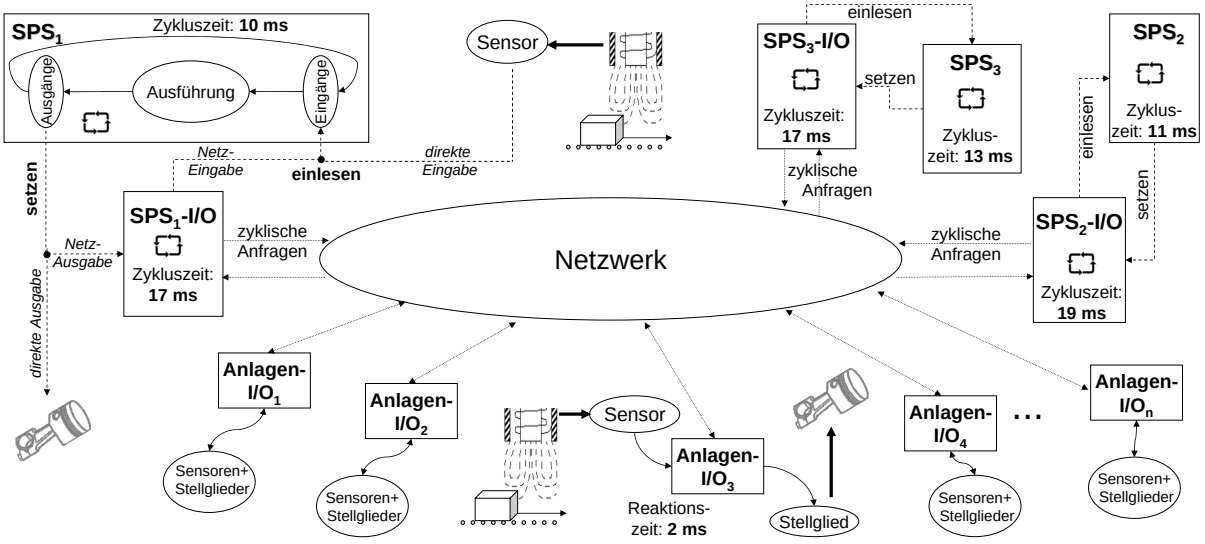

Abbildung 1: Netzwerkbasiertes Automatisierungssystem

Das in Abbildung 1 gezeigte NAS besteht aus drei SPSen, einer größeren Anzahl an Sensoren und Stellgliedern, sowie das die einzelnen Komponenten miteinander verbindende Netzwerk inklusive der zugehörigen Netzwerkschnittstellenhardware (I/O-Karten). Die traditionelle Möglichkeit, Sensoren und Stellglieder direkt an die SPS anzuschließen, ist zwar auch in modernen Anlagen noch vorhanden, soll hier jedoch nur als Vergleichsprozess berücksichtigt werden.

SPSen arbeiten zyklusbasiert, d.h. sie durchlaufen einen stets ähnlichen Ablauf, innerhalb welchem u.a. die Eingänge gelesen, die Ausgänge gesetzt und die Befehlszeilen abgearbeitet werden. Die Arbeitsweise der SPS-I/Os ist hingegen je nach Hersteller und Modell unterschiedlich ausgeprägt, wird im Rahmen dieser Arbeit jedoch ebenfalls als zyklisch und auf einer Client-Server-Arbeitsweise basierend, angenommen. Die Anlagen-I/Os stellen hierbei die Server dar, während die SPS-I/Os die Clients sind. Die Arbeitsweise der Anlagen-I/Os

ist daher ereignisgetrieben, d.h., sobald eine Anfrage eintrifft, wird diese bearbeitet und die Antwort an die anfragende SPS-I/O gesendet. Mit einer Anlagen-I/O sind i.d.R. mehrere Sensoren und/oder Stellglieder verbunden, wobei innerhalb einer Anfrage sämtliche Sensorwerte abgefragt und alle Stellglieder gesetzt werden können. Da es nun möglich ist, dass verschiedene SPSen auf ein und denselben Sensor oder zumindest auf Sensoren oder Stellglieder an ein und derselben Anlagen-I/O zugreifen, kann es zu Wartezeiten bei der Beantwortung der Anfragen kommen.

„Netzwerk“ ist ein abstrakter, die tatsächliche Architektur verbergender, Begriff. In der Praxis gibt es eine Vielzahl von Netzwerkarchitekturen und -protokollen, deren explizite Darstellung eine eigene Arbeit füllen würde. Zweifelsohne spielt das Netzwerk eine wichtige Rolle. Sein Anteil an den in einem NAS auftretenden Verzögerungen ist jedoch, wie in Kapitel 6 gezeigt wird, eher gering. Insofern ist es bedauerlich, dass sich akademische Untersuchungen meist nur auf Netzwerkaspekte konzentrieren und die durch die einzelnen Komponenten verursachten Verzögerungen als vernachlässigbar angenommen werden [Moyne und Tilbury, 2007].

Der derzeit wichtigste Netzwerktyp ist Ethernet. Eine Übersicht über industrielle Ethernetstandards wie z.B. PROFINET, Powerlink oder MODBus/TCP, findet sich in [Humphrey, 2006]. Ein wichtiges Kriterium für die Klassifizierung von Netzwerken ist deren Deterministik. Da Standard Ethernet nach [IEEE 802.3] dieses Kriterium nicht erfüllt, ist es nach Ansicht vieler Experten für sicherheitstechnische Anwendungen nicht geeignet, weshalb z.B. [Moyne und Tilbury, 2007] zwischen Netzwerken für Steuerungs-, Sicherheits- und Diagnoseaufgaben unterscheiden und unter diesem Aspekt unterschiedliche industrielle Netzwerke vorstellen. Für eine Übersicht der für die Sicherheitstechnik wichtigen Anforderungen an Ethernet-basierte Kommunikationssysteme wird auf [Glaser, 2007] verwiesen.

Um Ethernet deterministisch(er) zu gestalten, wurden verschiedene Ansätze unternommen, wobei die Mehrzahl zusätzliche Ebenen oberhalb von Standard Ethernet oder TCP/IP einzieht (vgl. z.B. [Felser, 2005]). Selbstverständlich können Netzwerke auch kabellos sein, was dann allerdings zu verschärften Anforderungen führt. In [Egea-Lopez et al., 2005] findet sich eine diesbezügliche Diskussion unterschiedlicher Technologien für den industriellen Einsatz sowie eine Liste von zu beachtenden Erfordernissen und Charakteristiken. Als Erfordernisse werden z.B. Skalierbarkeit, Flexibilität, Security und geringe Störanfälligkeit genannt und als Charakteristiken z.B. Kapazität, Reichweite, Betriebskosten, Umgebungsbedingungen und Lizenzfragen. Kriterien also, die in dieser Weise auch für jedes drahtbasierte Netzwerk eine Rolle spielen.

Innerhalb der Automatisierungstechnik wird das jeweilige Netzwerk in der Regel als gegeben vorausgesetzt und stattdessen das Gesamtsystem analysiert. Allerdings gibt es zwei verschiedene Forschungsrichtungen. Während bei NCS (Network Control Systems, [Walsh et al., 2002; Hespanha et al., 2007]), die Stabilitätsanalyse des kontinuierlichen Reglers im Mittelpunkt steht, liegt bei NAS der Schwerpunkt in der Verlässlichkeits- und Qualitätsanalyse. Und somit, wie im folgenden Abschnitt zu erläutern, implizit in der Analyse von Antwortzeiten. Daran anschließend führt Abschnitt 2.2 in die Antwortzeitanalyse und die

daraus resultierenden Fragestellungen ein. Hierauf aufbauend werden in Abschnitt 2.3 die für eine Analyse von NAS maßgeblichen Aspekte formuliert und in Abschnitt 2.4 literaturbekannte Analysemethoden und ihre Eignung zur Analyse von NAS dargestellt.

2.1 Verlässlichkeit und Qualität

Verlässlichkeit

Unter Verlässlichkeit versteht man ein mehrdimensionales Merkmal, welches sich messtechnisch als Menge unterschiedlicher Kenngrößen beschreiben lässt. Aufgrund der geringen Datenbestände wird die Verlässlichkeit jedoch i.d.R. modellanalytisch untersucht. Je nach Fachgruppe finden sich unterschiedliche Definitionen und Anforderungen. [DIN 40041] übersetzt das englische dependability mit Zuverlässigkeit und definiert diese als „die Fähigkeit einer Betrachtungseinheit, innerhalb der vorgegebenen Grenzen denjenigen durch den Verwendungszweck bedingten Anforderungen zu genügen, die an das Verhalten ihrer Eigenschaften während einer gegebenen Zeitdauer gestellt sind“. Nach [IFIP WG-10.4] ist dependability hingegen als „The trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers“ definiert.

Nach [Avižienis et al., 2004] gliedert sich dependability (Verlässlichkeit) in Zuverlässigkeit, Verfügbarkeit, Sicherheit (i.S.v. Safety), und Instandhaltbarkeit auf. Mathematische Definitionen von Zuverlässigkeit, Verfügbarkeit und Instandhaltbarkeit finden sich in der [IEC 61703] und Berechnungsvorschriften für Sicherheitswerte in [DIN EN IEC 61508-6].

Die Verlässlichkeit als Ganzes wird von [Avižienis et al., 2004] ebenso als Eigenschaft des Gesamtsystems geführt, wie Performanz, Funktionsfähigkeit, Security und Kosten. Ähnliches gilt für [Schnieder und Slovak, 2007], die zwischen Leistungsfähigkeits-, Qualitäts-, Verlässlichkeits- sowie Kostenaspekten unterscheiden und die Verlässlichkeit durch systembezogene (Verfügbarkeit, Sicherheit) und komponentenbezogene (Zuverlässigkeit, Instandhaltbarkeit) Eigenschaften spezialisieren.

All diese Größen stehen mit auf Anforderungen bezogenen Wahrscheinlichkeiten in Zusammenhang. Für die Verfügbarkeit könnte eine solche Anforderung beispielsweise lauten, dass ein System innerhalb einer vorgegebenen Zeit reagieren muss. Die Verfügbarkeit gibt dann den Anteil jener Fälle an, in denen das System diese Anforderung erfüllt. Die Bestimmung dieses Anteils führt auf die Bestimmung der Antwortzeitverteilung des betrachteten Systems.

Qualität

Im Zusammenhang mit Computernetzen findet sich als Gütemaß die so genannte Quality of Service (QoS). Diese auf generalisierten Faktoren wie Paketverluste, Bandbreite und Übertragungszeiten basierende Funktion ermöglicht den einfachen Vergleich und die effiziente Optimierung von Computernetzen. [Moyne und Tilbury, 2007] definieren die QoS eines

Netzwerks als „a multidimensional parameterized measure of how well the network performs this function“ und erläutern die QoS-Parameter Geschwindigkeit und Bandbreite sowie Delay und Jitter unter dem Aspekt „Balancing the Parameters“. Für eine Übersicht der QoS-Faktoren sei auf [Soucek und Sauter, 2004] verwiesen.

Im Bereich der klassischen kontinuierlichen Regelungstechnik findet sich dagegen das Gütemaß der Quality of Control („Regelgüte“). Dieses Maß basiert auf der Generalisierbarkeit der zugrunde liegenden Problembeschreibungen und ist i.d.R. quadratisch über diversen Eigenschaften wie z.B. der Stabilitätsreserve, der Konvergenzzeit oder dem Überschwingverhalten definiert. Die Verwendung einer solchen Qualitätszahl (q-factor) ist möglich, da sich die Problemstellung auf die generische Formulierung des Einregelns von Führungsgrößen und Ausregelns von Störgrößen bringen lässt.

Ein allgemeines Gütemaß für NAS ist hingegen aufgrund der stets anlagenspezifischen Steuerungsauslegung und der oftmals steuerungsspezifischen Aufgabenstellungen („halte nach Passieren des Signalgebers an“) nicht möglich, da sich das Problem selbst nicht generalisieren lässt. Durchaus möglich ist es jedoch, den Erfülltheitsgrad problemspezifischer Anforderungen zu bewerten und hieraus spezifische Gütemaße abzuleiten. Durch Abstraktion der jeweiligen Problembeschreibung auf eine Menge von Templates lassen sich darüber hinaus unterschiedliche Anforderungen kategorisieren. Mittels dieser Templates ist es dann möglich, eine Menge von Qualitätskriterien abzuleiten und eine NAS-orientierte Quality of Control Definition anzugeben:

Definition: Die Quality of Control (QoC) eines ereignisdiskreten Steuerungs-Systems (discrete event control system – DECS) ist als Leistungsvektor definiert. Dieses Qualitätsmaß beschreibt eine quantifizierbare Klassifizierung der Erreichbarkeit meist verbal definierter Eigenschaften. Mit letzteren sind hierbei spezifische Steuerungs-Eigenschaften gemeint, also Eigenschaften, die funktional an eine spezielle Steuerungs-Aufgabe, bzw. die Anforderungen einer speziellen Anlage, gebunden sind. Der Grad der Erreichbarkeit ([0...1]) wird auf der Basis eines reell-wertigen Vektors angegeben.

Im allgemeinen Fall kann dieser QoC-Vektor auf einer Menge von Qualitätskriterien basieren (z.B. Ausschuss, Mittelwerte, Verteilungen). Die Elemente des so erzeugten QoC-Vektors repräsentieren dann Qualitätserfülltheitsgrade verschiedener Teilgebiete (z.B. Genauigkeit, Ausschuss, Zeitquotienten). Obwohl es möglich ist, die einzelnen Vektorwerte in Relation zueinander zu stellen, macht es verhältnismäßig wenig Sinn, sie aufzuaddieren oder zu mitteln. Dies liegt auf der einen Seite daran, dass – im Gegensatz zur kontinuierlichen Regelungstheorie – keine generalisierbaren Werte (Regelgüte, Stabilitätsreserve, Einschwingzeit usw.) Verwendung finden können. Auf der anderen Seite machen eine Reihe von Diskontinuitäten und die oftmals komplett unterschiedliche Bedeutung der einzelnen Vektorelemente die Addition nicht begründbar. Die Suche nach einer optimalen Lösung besteht daher in der Regel im Finden einer Lösung, welche die gegebenen Randbedingungen erfüllt.

Zusammenfassend bedeutet dies, dass sich die Qualitätsaspekte eines NAS auf Performanzaspekte zurückführen lassen, bzw. aus der Kenntnis der Systemperformanz auf die QoC eines DECS geschlossen werden kann. Wichtige Aspekte der Systemperformanz lassen sich jedoch aus der Antwortzeitverteilung extrahieren. Dies ermöglicht dann beispielsweise Aussagen über die für den Endkunden sichtbare „Qualität“ des Produktes einer automatisierten Anlage, oder über die produktions- und kostentechnisch wichtige Frage des Ausschusses, der benötigten Zeiten und der Wiederholungs- sowie Fertigungsgenauigkeit. Ein ausführliches Beispiel hierzu findet sich in [Greifeneder und Frey, 2006b].

2.2 Antwortzeitanalyse

Zur Einstufung eines automatisierungstechnischen Systems als zuverlässig, können Anforderungen wie z.B. „*die Reaktion auf einen Signalwechsel erfolgt innerhalb von 200 ms*“ oder „*ein lediglich 5 ms lang anliegendes Signal wird von der Steuerung registriert*“ dienen. Während sich Anforderungen dieser Art im Falle eines klassischen, direkt mit der Strecke verdrahteten, Reglers i.d.R. als Funktion der Prozessortaktfrequenz angeben lassen, ergibt sich selbst im Falle einer einzigen zyklisch ihre Eingänge abfragenden SPS ein ganzes Spektrum (im einfachsten Fall gleichverteilter) möglicher Verzögerungszeiten. Bereits beim Auftreten zweier unabhängiger (d.h. unterschiedlicher oder nicht exakt synchronisierter) Zyklen (z.B. SPS und zugehörige Interfacekarte) wird das resultierende Verzögerungsspektrum komplex. Sind mehrere Prozesse gleichzeitig aktiv, wird eine exakte Synchronisation der einzelnen Komponenten immer unwahrscheinlicher, wobei Zugriffskonflikte und Wartezeiten ihr Übriges tun. Würde man von einem solchen System erwarten, dass es obige Spezifikationen grundsätzlich erfüllt (worst-case Analyse), dann würde dies zu kaum erfüllbaren Anforderungen an die Hardware-Struktur führen, und somit die Verwendung von NAS ad absurdum führen. Allerdings ist es im Normalfall gar nicht notwendig, diese Spezifikationen grundsätzlich zu erfüllen, sondern es genügt, wenn sie mit großer Sicherheit erfüllt werden. Obige Spezifikationsanforderungen können daher z.B. zu „*die Reaktion auf einen Signalwechsel erfolgt in 99,9% der Fälle innerhalb von 200 ms*“ oder „*ein lediglich 5 ms lang anliegendes Signal wird mit 90%iger Wahrscheinlichkeit von der Steuerung registriert*“ abgeschwächt werden.

Während im Bereich des Softwareengineerings zwischen funktionalen und nicht-funktionalen Korrektheitseigenschaften unterschieden wird – wobei die Zeit zu den nicht-funktionalen Eigenschaften gezählt wird, unterscheidet man in der Echtzeitdomäne zwischen wertbasierten und zeitbasierten Eigenschaften ([Kopetz, 2003]). Dies bedeutet, dass zunächst geprüft werden muss, ob das System die korrekte Funktion aufweist (wertbasierte Korrektheit). Ist dem so, stellt sich die Frage, ob die Funktionalität auch in der zulässigen Zeit erbracht wird (zeitbasierte Korrektheit). Dieser temporale Bereich ist insbesondere auch deshalb interessant, weil die Prüfung wichtiger Verlässlichkeitseigenschaften darauf beruhen. Da es in der AT kaum Anwendungsbeispiele gibt, bei denen beliebige Verzögerungen möglich oder zulässig wären, werden hier nur die zeitabhängigen Anforderungen an ein AT-System betrachtet.

Das durch die Vernetzung und die dezentrale, i.d.R. zyklusbasierte, Ausführung gekennzeichnete System weist eine Superposition aus konstanten und zyklischen Verzögerungen auf. Stochastische Ausfälle von Komponenten, Fehler bei der Kommunikation und durch die Mehrbenutzerumgebung hervorgerufende Wartezeiten führen zu weiteren Verzögerungen und resultieren in unterschiedlichen Fragestellungen, welche im Folgenden genauer diskutiert werden.

Für die Analyse von **Antwortzeiten** eines NAS ist es erforderlich, den in Abbildung 2 beispielhaft gezeigten Prozess zu berücksichtigen, welcher sich aus dem Erkennen des Signalwechsels, sowie der zugehörigen Verarbeitung und anschließenden Reaktion an einem Stellglied, zusammensetzt. Ein solcher Ablauf beginnt mit dem Versenden einer Sensorwertanfrage der SPS-I/O an die Anlagen-I/O. Nach der Netzpassage, der Bearbeitung durch die Anlagen-I/O und einer weiteren Netzpassage, wird diese Antwort von der SPS verarbeitet und eine zugehörige Stellgliedanweisung (i.d.R. zusammen mit der nächsten Sensorwertanfrage) von der SPS-I/O über das Netzwerk zur Anlagen-I/O geschickt. Der Ablauf endet mit der Aktivierung des Stellgliedes.

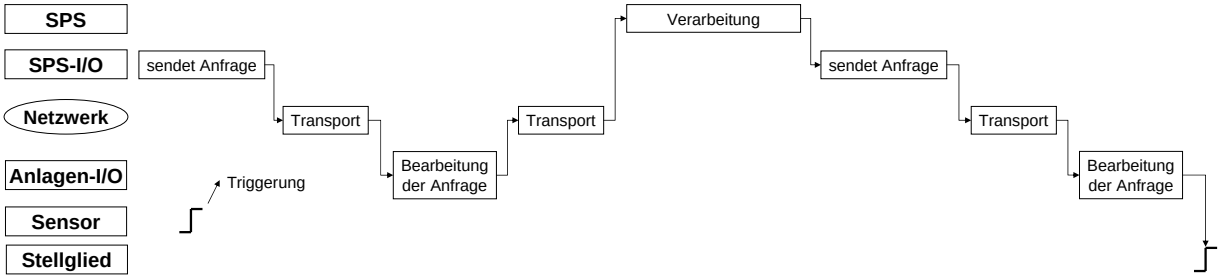

Abbildung 2: Antwortzeit eines NAS

Da ein Sensorwert von der Anlagen-I/O nur ausgelesen werden kann solange er anliegt (bzw. durch eine zwischengeschaltete Logik gehalten wird), hat diese Signalaktivitätsdauer direkten Einfluss auf die Fragestellung der Analyse. In einem automatisierungstechnischen System lassen sich die folgenden drei Signaltypen unterscheiden (Abbildung 3):

I Ein Signal, das seinen Wert beibehält, bis es vom Controller erkannt wird. Ein Beispiel dieses Signaltyps ist der Notaus. Ist er einmal ausgelöst worden, wird er erst wieder zurückgesetzt, nachdem die Information erhalten, verarbeit und eine entsprechende Aktion ausgelöst worden ist. Die Frage in diesem Fall lautet daher: „Wie lange dauert es, bis das Signal erkannt wird?“

II Ein Signal, das nur für eine bestimmte Zeit τ_{fixed} aktiv ist. Ist diese Zeit abgelaufen, so ist die Information unwiderruflich verloren, falls sie nicht vorher ausgelesen worden ist. Die Frage in diesem Fall lautet daher: „Wird das Signal überhaupt erkannt?“, bzw. „Wie groß muss τ_{fixed} mindestens sein, damit das Signal mit einer gegebenen Wahrscheinlichkeit erkannt wird?“

III Die Kombination der vorherigen beiden Fälle. Hierunter fällt z.B. ein Signal, welches sich mit einer bestimmten Wahrscheinlichkeit ändern kann und anschließend bis zum nächsten Signalwechsel, mindestens jedoch für eine Zeitspanne von τ_{min}, angezeigt

wird. Daraus resultieren zwei Fragen: „Kann der Signalwechsel erkannt werden, bevor sich ein erneuter Wechsel einstellt?“ und „Wie lange dauert es, bis er erkannt wird?“

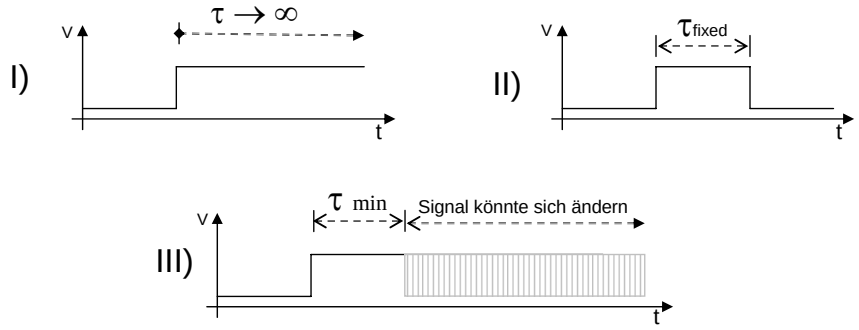

Abbildung 3: Drei verschiedene Typen von Eingangssignalen

Neben dem Eingangsverhalten müssen Bearbeitungs- und Wartezeiten ebenso berücksichtigt werden, wie Ausfälle und Fehler. Zur Modellierung letzterer reicht die alleinige Kenntnis der zugehörigen Auftrittswahrscheinlichkeit nicht aus. Ebenso wichtig ist es, die Zeiten zu berücksichtigen, die notwendig sind, um eine solche Situation zu erkennen, sowie, um sie zu beheben. Außerdem muss die Wahrscheinlichkeit dessen erwogen werden, dass eine Information vollständig verloren geht, d.h. Berücksichtigung der Wahrscheinlichkeit, dass eine notwendige Information nicht innerhalb eines gesetzten Zeitfensters eintrifft.

Hieraus lassen sich die für die Analyse (von Antwortzeiten) eines NAS grundlegenden Fragestellungen ableiten und in die folgenden vier Untergruppen aufteilen:

1. *Maximal- und Minimalzeiten*: Beispiele hierfür sind die Antwortzeit, welche ab der Aktivierung des Notausknopfes vergeht sowie das früheste zulässige Öffnen einer Klemme nach Auslösen eines Befehls. Die zugehörigen Fragestellungen lauten:
   – Ab wann kann mit einer Reaktion gerechnet werden?
   – Wann hat sich eine Reaktion spätestens eingestellt?

2. *Verteilungen*: Zur Analyse von Performanz und Qualität ist es unerlässlich, die komplette Antwortzeitverteilung ermitteln zu können. Auf dieser Basis lassen sich dann Mittelwerte und Streuungen ebenso berechnen, wie z.B. Fehlersensibilitäten. Verteilungen lassen sich entweder analytisch bestimmen oder durch wiederholte Fragestellung nach dem auf einen Zeitabschnitt entfallenden relativen Anteil.

3. *Intervallwahrscheinlichkeiten*: Hierunter fallen alle auf wahrscheinlichkeitsbasierten Grenzlinien definierten Fragestellungen, wie z.B.: Mit welcher Wahrscheinlichkeit kann auf ein Signal innerhalb eines gegebenen Zeitfensters reagiert werden? Welche Zeit ist zu gewähren, um einen gegebenen Prozentsatz aller Antworten abzudecken? Mit welcher Wahrscheinlichkeit liegt die Antwortzeit außerhalb vorgegebener Grenzen?

   Hinweis: Zur Beantwortung derartiger Fragestellungen genügt es, einzelne Werte der Antwortzeitverteilungsfunktion zu bestimmen. Die Berechnung der kompletten Antwortzeitverteilung ist daher nicht notwendig (wohl aber hinreichend).

4. *Differenzen, Abstände*: Zeitdifferenz mit der zwei aufeinander folgende Datenpakete eintreffen oder die Frage nach der Wahrscheinlichkeit einer korrekten Reihenfolge beim Eintreffen dieser Datenpakete. Hierzu zählt z.B. die Frage nach der Zeitdifferenz zweier, von zwei nacheinander aufgetretenen Eingangssignalwechseln erzeugten, Ausgangssignale oder die Frage nach der Wahrscheinlichkeit, dass dieser Abstand über einem vorgegebenen Grenzwert liegt.

2.3 Anforderungen an die Analysemethodik

Für eine Analyse von (auf Antwortzeiten zurückgeführten) Verfügbarkeits- und Performanzfragen eines NAS ist es somit erforderlich, eine Methodik zu wählen, welche die folgenden fünf Punkte erfüllt:

1. *Modellierungsmöglichkeit* von stochastischem und zeitbewertetem Eigenverhalten aller Komponenten. Hierbei ist insbesondere auf die Notwendigkeit hinzuweisen, sowohl beliebige Zeitverteilungen als auch deterministische und stochastische Entscheidungen implementieren zu können.
2. *Definitionsmöglichkeit* beliebiger stochastischer und zeitbewerteter Anfangs- und Randbedingungen. Dies ist u.a. deshalb erforderlich, da System und Beobachtung i.d.R. nicht synchronisiert sind.
3. *Prüfbarkeit* von stochastischen und zeitbehafteten Eigenschaften. Wichtig ist es, Wahrscheinlichkeitsverteilungen über der Zeit und Wahrscheinlichkeiten für ein Verhalten innerhalb bestimmter Zeitbegrenzungen ermitteln zu können. Reine Extremwertanalysemethoden sind hingegen aufgrund des stochastischen Eigenverhaltens von NAS nicht geeignet.
4. *Implementierbarkeit von Abhängigkeiten* sowohl zwischen einzelnen Komponenten als auch zwischen Komponenten und Eingangssignalen. Dies ist zur Beschreibung der diversen in einem NAS auftretenden Synchronisationsaspekte und Prozessabfolgen unerlässlich.
5. *Statistische Belastbarkeit* der Ergebnisse. Da in einem NAS viele Effekte nur mit geringer Wahrscheinlichkeit auftreten, ist es notwendig, dass Vollständigkeit und Ergebniskonfidenz gefordert werden. Vollständigkeit bedeutet, dass alle Evolutionsmöglichkeiten des Systems auch wirklich in der Analyse berücksichtigt worden sind. Ergebniskonfidenz hingegen bedeutet, dass die so erzielten Ergebnisse als akzeptable (i.S. der Fehlergenauigkeit) Näherung der tatsächlichen Wahrscheinlichkeitsverläufe angesehen werden können.

Neben diesen fünf Ausschlusskriterien ist insbesondere die Skalierbarkeit des der Analysemethodik zu übergebenden Systemmodells zu nennen, da nur so Rekonfigurationsuntersuchungen, wie z.B. das Hinzuklemmen einer zusätzlichen Komponente, mit vertretbarem Aufwand und in akzeptabler Zeit lösbar sind. Darüber hinaus ist es vorteilhaft, wenn der, sich auf Zeit und Hauptspeicherbedarf beziehende, Ressourcenverbrauch des Rechners möglichst niedrig ist und es unterstützende Softwaretools gibt.

2.4 Antwortzeitanalysemethoden

Die bekannten Verfahren zur Bestimmung von Antwortzeiten unterscheidet man in messtechnische, analytische, statische, simulative und verifikationsformale Ansätze. Hierbei spielen zwei in der Netzanalyse und der Automatisierungstechnik unterschiedlich belegte Begriffe eine Rolle, nämlich Zyklus- und Antwortzeit. In der Netzanalyse versteht man unter der (Netzwerk-)Zykluszeit (Network Cycle Time) jene Zeit, welche von dem Zeitpunkt vergeht, zu dem ein bestimmter Knoten des Netzes senden durfte, bis zu dem Zeitpunkt, an dem er dies erneut darf. Unter der Antwortzeit wird in der Netzanalyse jene Zeitdauer verstanden, die vom Versenden eines Paketes durch einen Client zum Server bis zur Ankunft der Antwort des Servers beim Client vergeht (z.B. ping). In der Automatisierungstechnik versteht man hingegen unter Antwortzeit jene Zeitdauer, welche vom Eintritt eines Ereignisses bis zur vollständigen Ausführung einer oder aller zugehörigen Reaktionen benötigt wird. Der Begriff der Zykluszeit findet sich in der Automatisierungstechnik i.d.R. auf SPS oder (Netz-)Interfacekarten bezogen.

Obige Begrifflichkeiten berücksichtigend, werden im Folgenden verschiedene, literaturbekannte Ansätze zur Antwortzeitbestimmung vorgestellt und ihre Eignung zur Analyse von Antwortzeiten in NAS entsprechend der im vorherigen Abschnitt dargestellten Kriterien beurteilt.

Messtechnische Verfahren

In der Literatur findet sich eine ganze Reihe von Messaufbauten und vor allem Dokumentationen zu durchgeführten Messungen für die verschiedensten Konfigurationen. Die Mehrzahl beschränkt sich jedoch auf den reinen Netzwerkaspekt und die Optimierung der zugrundeliegenden Protokolle für verschiedene Anwendungen. Beispielhaft seien genannt, [Irey et al., 2000], für Glasfaserübertragungen, [Parrott et al., 2006], für UDP mit OPC und VPN Verkehr, [Cena et al., 2004] für einen methodischen Messansatz zur Latenzbestimmung im Ethernet, sowie [Rauchhaupt et al., 2007], für einen methodischen Messansatz zur Bestimmung von Zuverlässigkeiten industrieller Funklösungen.

Für die Reihe der Messungen zu NAS-spezifischen Fragestellungen wird stellvertretend auf [Jasperneite, 2001, 2002] und [Fritsch et al., 2006] verwiesen. Während sich Jasperneite u.a. mit Untersuchungen zum Einfluss der Zykluszeit der SPS befasste, untersuchten Fritsch et al. die Abhängigkeit der Antwortzeit von der Häufigkeit eines angelegten Signalwechsels, wobei die SPS einen festen Zyklus besitzt und die verwendete UDP-Übertragung als für AT-Zwecke ausreichend zuverlässig eingestuft wird.

Ein wichtiger Aspekt messtechnischer Ergebnisse ist die Möglichkeit, diese zur Validierung und insbesondere zur Parametrierung der jeweiligen Modelle in den anderen Ansätzen zu verwenden. Beispielsweise sei die, einen simulativen und einen verifikationsformalen Ansatz mit messtechnischen Ergebnissen vergleichende, Arbeit von [Greifeneder et al., 2007] genannt.

Analytische und statische Verfahren

Statische Analysemethoden liefern Aussagen, die aufgrund der Konfiguration des Systems getroffen werden können. Das heißt, alle Prozesse im System werden quasi eingefroren und durch charakteristische Werte repräsentiert, wie z.B. den Minimal-, den Maximal- oder den Mittelwert. Während es bei der Verwendung von Mittelwerten meist um prinzipielle Fragen der Kapazitätsauslegung geht, ist die Analyse mittels Minimalwerten ebenso ein Sicherheitsaspekt („Frühest zulässiges Öffnen einer Sicherheitsklemmvorrichtung“), wie die mittels Maximalwerten („Maximale Verzögerung eines Notaussignals“). Die maximalwertbasierte Analyse wird darüber hinaus für detailliertere Kapazitätsauslegungsfragen verwendet. Beispielhaft sei auf [Pinho und Vasques, 2000] verwiesen, welche verschiedene Protokolle zur optimalen Neuübertragung bzw. Duplizierung von Meldungen auf CAN mittels einer statischen Worst-Case-Methodik untersuchten. Die bekannteste Worst-Case-Methodik ist indes der Network Calculus, welcher z.B. von [Stanczyk und Obuchowicz, 2003] auf nebenläufige Prozesse oder von [Zhang et al., 2006] zur Bestimmung des maximalen Closed-loop control delay in switched industrial ethernet, verwendet wurde. Darüber hinaus finden sich in [Schmitt und Zdarsky, 2006], neben der Beschreibung einer Software für Network Calculus, auch einige Beispielanalysen für Verzögerungen in einem drahtlosen Sensornetzwerk.

Der größte Vorteil der statischen Analysemethoden liegt in ihrer Schnelligkeit und der Tatsache, dass auch komplexe (i.S.v. große) Systeme verhältnismäßig einfach und in vertretbarer Zeit analysiert werden können. Nachteilig – und ein Ausschlusskriterium für die Anwendung auf NAS – ist hingegen, dass keine Wahrscheinlichkeitsverteilungen bestimmbar sind.

Dies leitet zu den analytischen Ansätzen über. Hierbei werden die einzelnen Komponentenverhaltensmodi unter der Annahme linearer Unabhängigkeit überlagert. Als Ergebnisse sind somit Mittelwerte und Verteilungen möglich. Während sich [Schnieder und Kraft, 1980] bei der analytischen Berechnung der Antwortzeiten sich überlagernder (unabhängiger) Bearbeitungszyklen klassischer Steuerungssysteme auf Mittel- und Extremwerte beschränkten, gibt es eine große Zahl von Anwendungsbeispielen, die mittels analytischer Ansätze Verteilungen erzeugten. Beispielhaft seien [Dingle et al., 2002] und [Cena et al., 2007] genannt: [Dingle et al., 2002] transformieren einen mittels verallgemeinerter stochastischer Petri-Netze (GSPN) erzeugten Satz linearer Gleichungen in die Laplace-Ebene, in welcher sich die als unabhängig angenommenen Dichtefunktionen direkt addieren lassen. Die Rücktransformation liefert anschließend die gewünschte Antwortzeitverteilung. [Cena et al., 2007] untersuchen die aus der Addition von konstantem Jitter mit der aus der Faltung von Übertragungs- und Wartezeitverteilungen entstehenden Verteilung resultierende Reaktionszeit einer WLAN-Übertragung. Die auf einer selbstentwickelten Messplattform durchgeführte Validierung kommt zu einer hohen Übereinstimmung von analytischer und messtechnischer Lösung.

Eine interessante Variante stellt die von [Vojnović und Boudec, 2002] eingeführte stochastische Erweiterung des Network Calculus (NC) dar, mittels derer sich Verteilungen

ermitteln lassen, falls die Eingangssignale linear unabhängig voneinander sind. [Ridouard et al., 2007] untersuchen hiermit die Übertragungszeit eines AFDX-Netzwerks, wie es in der Luftfahrtindustrie eingesetzt wird. Die Autoren vergleichen ihre Ergebnisse auch mit simulativen und mittels klassischem NC erhaltenen Resultaten. Als Vergleichsgröße dient die Zeit, innerhalb welcher 99,99% aller Übertragungen erfolgen. Die größten Werte liefert erwartungsgemäß der Worst-Case des klassischen NC. Die geringsten, die Simulation. Die Erklärung für letzteres bleiben die Autoren jedoch schuldig.

Nicht zuletzt ist die Warteschlangentheorie zu nennen, mit welcher sowohl Mittel- und Extremwerte, als auch Verteilungen erstellt werden können, falls die einzelnen Komponenten linear unabhängig sind. So nutzen z.B. [Miorandi und Vitturi, 2004] die Warteschlangentheorie zur analytischen Berechnung der Netzwerkperformanz unter Verwendung von Paketverlustwahrscheinlichkeiten und vergleichen die so erhaltenen Mittelwerte von Jitter und Latenzzeiten mit Messungen in einem drahtlosen Netzwerk. [Song et al., 2002] hingegen erzeugen mittels Warteschlangentheorie Extremwerte und Verteilungen, welche zur Charakterisierung von Verzögerungen in Switched Ethernet verwendet werden.

Obwohl es also analytische Methoden gibt, mittels derer sich exakte Ergebnisse erzielen lassen, stellt die Annahme unabhängiger Eingangsprozesse ein Ausschlusskriterium für die Analyse von NAS dar. Hiermit kann zwar der gemeinsame Gebrauch von Ressourcen, wie z.B. des Netzwerks, modelliert werden, nicht jedoch Abhängigkeiten und Prozessabfolgen.

Simulative Verfahren

Die simulativen Ansätze unterscheiden sich von den analytischen insbesondere durch die erweiterten Abbildungsmöglichkeiten: Dynamik, Abhängigkeiten und zufällige Entscheidungen lassen sich mit Hilfe von Simulationen nachbilden. Die dabei zum Einsatz kommenden Tools (vgl. hierzu auch [Årzén und Cervin, 2005]) sind jedoch entweder der Netzanalyse entliehen (z.B. OMNet++ [Varga, 2001]) oder verwenden Simulatoren ereignisdiskreter (z.B. CPNtools [Ratzer et al., 2003], PowerDEVS [Kofman et al., 2003]) oder kontinuierlicher Systeme (z.B. Matlab in Verbindung mit TrueTime [Ohlin et al., 2006], Dymola [Brück et al., 2002]). Ein positiver Aspekt ist, dass die Mehrzahl der Tools über Bibliotheken für diverse Netzwerkprotokolle verfügt.

Anwendungsbeispiele gibt es zur Genüge. So untersucht z.B. Juanole Quality of Service Parameter von kontinuierlichen Regelstrecken unterlagerter Netzwerke mittels TrueTime, wobei stochastische zeitbewertete Petri-Netze (STPN) zur Modellierung Verwendung finden (z.B. [Juanole und Mouney, 2005] für den CAN-Bus). Auch [Zimmermann und Trowitzsch, 2006] verwenden stochastische Petri-Netze zur Modellierung. Unter Verwendung einer hauseigenen Simulationssoftware (TimeNET) werden hierbei Leistungsmaße des neuen europäischen Zugleitsystems bestimmt. OMNet++ findet z.B. bei [Pereira et al., 2004] Anwendung, die ein sehr detailliertes NAS-Modell verwenden und die erhaltenen Ergebnisse mit einem analytischen Worst-Case Ansatz vergleichen, wobei der simulative Ansatz erwartungsgemäß deutlich besser abschneidet. Schließlich sind die simulativen NAS-Analysen in

[Jasperneite und Neumann, 2001] zu nennen, die dem Netzwerk einen geringeren Einfluss auf die Antwortzeit attestieren als den Speicherprogrammierbaren Steuerungen (SPS).

Eine sehr detaillierte Modellierung der NAS-Komponenten und -abläufe findet sich in [Marsal et al., 2006], welche das auf farbigen Petrinetzen (CPN) beruhende Simulationstool CPNtools der University of Aarhus verwenden. Das Besondere daran ist, dass sowohl die Komponenten, als auch die Nachrichten als Token eines High-Level-Petrinetzes dargestellt werden. Der Ansatz erlaubt es daher, qualitative Konfigurationsvergleiche verschiedener NAS-Architekturen in vertretbarer Zeit zu erstellen, hat allerdings den Nachteil, dass das abstrakte Modell für den Laien schwer verständlich ist.

In [Liu und Frey, 2007] wird ein Weg begangen, der auch den Einbau kontinuierlicher Komponenten ermöglicht. Hierbei werden die Modelle in Modelica modelliert und dann in Dymola grafisch verbunden. Dymola hat den Vorteil, dass seine komponentenorientierte Bedienoberfläche im AT-Bereich leicht verständlich ist. Das Bibliothekskonzept von Modelica ermöglicht darüber hinaus die schnelle Erstellung von Modellen, nicht zuletzt auch, da die Autoren eine Tool-Box für den AT-Bereich zur Verfügung stellen wollen, welche Modelle für standardisierte Netzwerkkomponenten und Prozesshardware enthält. Die vorgestellte Methodik eignet sich besonders gut, wenn die Auswirkungen von Netzwerk und Steuerung auf den geschlossenen Kreis untersucht werden sollen.

Bei allen simulativen Ansätzen besteht jedoch die Vollständigkeits- und die Konfidenzproblematik: Will man mittels simulativer Ansätze ein akzeptables Ergebnis erreichen, so ist es notwendig, die Anfangsverteilung intelligent zu wählen und eine ausreichende Anzahl an Durchläufen zu vollführen. Das Problem der Anfangsverteilung liegt darin, dass sichergestellt werden muss, dass tatsächlich alle theoretisch möglichen Anfangskombinationen mit genau der Häufigkeit auftreten, mit der sie im Systemmodell vorgesehen sind. Es folgt direkt, dass die Anzahl der Durchläufe entsprechend groß sein muss, um im Mittel auch kleine Auftretenswahrscheinlichkeiten sowohl als Eingangsgröße (z.B. Fehler) als auch im Ergebnis erfassen zu können. Das grundsätzliche Problem aller simulativen Ansätze ist also, dass die ermittelten Ergebnisse weder eine Vollständigkeits- noch eine Konfidenzgarantie aufweisen; d.h., das aus der Überlagerung einer großen Anzahl von Simulationsläufen gewonnene Bild kann nur für sehr häufige Simulationsläufe, als in Bezug auf die korrekte Lösung asymptotisch, nicht jedoch als exakt, betrachtet werden. Die benötigte Anzahl der Simulationsläufe stellt eine Funktion der kleinsten theoretisch auftretenden Wahrscheinlichkeit dar und lässt sich mittels statistischer Ansätze der Konfidenzintervallschätzung bestimmen [Stöhr, 2007]. In Abbildung 4 ist beispielsweise der notwendige Stichprobenumfang (Ordinate) für eine, die relative Häufigkeit (Abszisse) betreffende Aussage (Ergebnis) dargestellt, wobei die Intervallbreite B als Parameter Verwendung findet und von einem Konfidenzniveau von 99% ausgegangen wird. Dies bedeutet, dass eine simulierte relative Häufigkeit r nur dann mit einer Wahrscheinlichkeit von 99% innerhalb des durch $[r - B, r + B]$ gegebenen Intervalls liegt, wenn mindestens n Simulationsdurchläufe getätigt werden. Die tatsächliche relative Häufigkeit liegt also mit einer Wahrscheinlichkeit von 99% innerhalb dieses Intervalls.

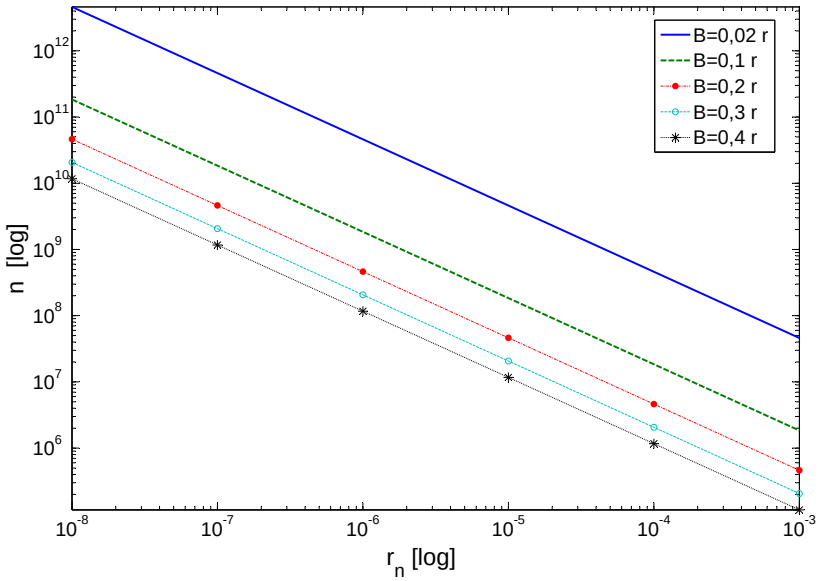

Abbildung 4: Notwendiger Stichprobenumfang, damit eine ermittelte relative Häufigkeit r mit 99%-iger Wahrscheinlichkeit im Intervall [r-B,r+B] liegt [Stöhr, 2007].

Deutlich wird, dass schon für relative Häufigkeiten von 10^{-4} fast 20 Millionen Simulationsdurchläufe notwendig sind, wenn mit 99%-iger Wahrscheinlichkeit der tatsächliche Wert innerhalb von ±10% liegen soll. Dies relativiert den oftmals angenommenen Geschwindigkeitsvorteil der simulationsbasierten Ansätze deutlich, da gerade in NAS durch das dynamische Zusammenspiel vieler Komponenten und die Existenz von Ausfällen, Interferenzen und Störungen eine Vielzahl von Situationen mit kleinen Auftrittswahrscheinlichkeiten existieren (vgl. [Greifeneder et al., 2007]).

Verifikationsformale Verfahren

Verifikationsformale Ansätze haben den Anspruch, alle praktisch möglichen Evolutionen des Systems abzudecken. Hierzu sind nicht nur streng formale Modelle, sondern auch eine spezielle Abfragelogik notwendig, welche zur Formulierung der interessierenden Systemeigenschaften Verwendung findet. Einen guten Überblick über die formalen Verifikationsmöglichkeiten zeitbewerteter Systeme gibt [Wang, 2004].

Bei der, in Kapitel 3 näher zu erläuternden, klassischen Modellverifikation (Model Checking, MC) steht die Frage im Mittelpunkt, ob eine zu definierende Eigenschaft eintreten kann (bzw. es möglich ist, dass sie nicht eintritt). Obwohl die zu überprüfenden Eigenschaften durchaus komplex sein können, ist das Ergebnis der MC stets boolescher Natur, d.h. MC ermöglicht Worst-Case-Analysen, jedoch keine Verteilungen. Im Vergleich mit den meisten statischen Ansätzen hat MC den Vorteil, dass paralleles, nicht-deterministisches, abhängiges und dynamisches Verhalten berücksichtigt und analysiert werden kann. Nachteilig wirkt sich aus, dass die Modellierung des Systems und die Formulierung der zu prüfenden Eigenschaften verhältnismäßig schwierig sind, da u.a. gewisse Modellierungsregeln zur

Eindämmung der Zustandsraumexplosion berücksichtigt werden sollten. Einige Beispiele hierzu finden sich z.B. in [Witsch und Vogel-Heuser, 2006].

Kontinuierliche Dynamik kann bei Verwendung der MC ebensowenig implementiert werden wie stochastisches Verhalten. Die Anwendungsbereiche der MC liegen daher im Nachweis spezieller, besonders kritischer Eigenschaften und der Verifikation reduzierter Modelle. So verwenden beispielsweise [Ermont et al., 2006] Model-Checking und das Softwaretool UPPAAL zur Worst-Case-Analyse der Netzübertragung in CAN, während [Hédia et al., 2005] eine Untersuchung von minimaler und maximaler Verzögerung sowie Verlustraten (QoS-Parameter) in Abhängigkeit eines Geräte-Treibers innerhalb einer Echtzeitanwendung beschreiben, wobei der Model-Checker IF zur Anwendung kommt. [Vyatkin und Hanisch, 2003] verwenden MC zur Funktionsvalidierung von flexiblen, durch verteilte Steuerungen gesteuerte, Fertigungssysteme. Dabei liegt das Ziel jedoch lediglich in der Frage, ob die Systemprozesse zeitlich zusammenpassen und nicht in der Analyse von Antwortzeiten.

Das eigentliche Manko der MC, nur Worst-Case-Analysen durchführen zu können, wird behoben, wenn man stattdessen die stochastische Erweiterung der MC, namentlich die wahrscheinlichkeitsbasierte Modellverifikation (Probabilistic Model Checking, PMC) verwendet. Hierfür können nicht nur Modelle mit wahrscheinlichkeitsbehafteten und zeitbewerteten Übergängen verwendet, sondern auch die Ergebnisse wahrscheinlichkeits- und zeitbasiert ausgegeben werden. Anders als bei simulationsbasierten Ansätzen ist ein simples mehrfaches Durchlaufen des Modells beim PMC weder möglich noch nötig, wobei der Vorteil der MC, statistisch belastbare Ergebnisse zu erhalten, selbstverständlich ererbt wurde. Näheres zur PMC findet sich in Kapitel 3, das diese Analysemethodik im Detail erläutert.

Methodenvergleich

Bezugnehmend auf die in Abschnitt 2.3 formulierten Anforderungen an eine Analysemethodik zeigt Tabelle 1, wie die einzelnen Methoden einzuordnen sind. Die statischen und die analytischen Ansätze scheitern leider daran, dass sie NAS-Systeme nicht abbilden können, da eine Unabhängigkeit der Komponenten vorausgesetzt wird, oder sie nur Extrem- bzw. Mittelwerte zurückliefern. An letzterem Kriterium scheitert auch die klassische Modellverifikation. Messtechnische Ansätze schließlich lassen sich zur Validierung und Parametrierung anderer Methoden nutzen, bleiben aber aufgrund ihres Hardware- und Zeitbedarfs auf die Bewertung existenter Lösungen beschränkt.

Bleibt ein Vergleich zwischen Simulation und wahrscheinlichkeitsbasierter Modellverifikation. Für die Simulation spricht, dass es eine große Zahl an Anwendungssoftware und insbesondere diverse Bibliotheken gibt, welche die Programmierung der zu simulierenden Systeme verhältnismäßig einfach und intuitiv machen. Hiervon kann im Falle von PMC leider nicht die Rede sein, was jedoch in erster Linie daran liegt, dass es sich um ein relativ junges Verfahren handelt, welches sicherlich weiterentwickelt und durch entsprechende grafische Softwareplattformen unterstützt werden wird.

Tabelle 1: Methodenvergleich auf Basis der in Abschnitt 2.3 formulierten Anforderungen. „√“ steht hierbei für *erfüllt*, „–“ für *nicht erfüllt* und „teilw.“ für *von manchen Vertretern dieser Gruppe erfüllt*.

	statisch	analytisch	MC	Simulation	PMC
Modellierungsmöglichkeit	–	teilw.	–	√	√
Definitionsmöglichkeit	–	teilw.	–	√	√
Eigenschaftsprüfbarkeit	–	teilw.	–	√	√
Impl. von Abhängigkeiten	–	–	√	√	√
Statistische Belastbarkeit	√	√	√	begrenzt	√

Ein kleiner Makel der PMC im Vergleich zur Simulation ist die Komplexitätsbeschränkung, welche durch die Größe des verifizierbaren Zustandsraumes bedingt ist. Aufgrund der Universalität des Ansatzes ist es jedoch möglich, große Systeme auf eine Menge abstrakter, repräsentativer und kritischer Modelle zu reduzieren und so die Beschränkungen von PMC auf kleine und mittlere Systeme elegant zu umgehen.

Als wichtigster Unterschied verbleibt somit, dass PMC unter Garantie sämtliche Möglichkeiten entsprechend ihrer Auftretenswahrscheinlichkeit berücksichtigt (Vollständigkeits- und Konfidenzkriterium) und somit exakte Ergebnisse liefert, während für simulative Ansätze auch mit sehr vielen Durchläufen lediglich asymptotische Ergebnisse erzielbar sind. Da PMC den simulativen Ansätzen hierdurch klar überlegen ist, wurde PMC als optimale Untersuchungsmethodik für die Antwortzeitanalyse in NAS ausgewählt.

3 Probabilistic Model Checking (PMC)

Die, von Wissenschaftlern aus Frankreich [Quielle und Sifakis, 1981] und den USA [Clarke und Emerson, 1981] unabhängig voneinander entwickelte (klassische) Modellverifikation (model checking, MC) ermöglicht es, Verhaltenseigenschaften eines Systems mittels eines automatisierten Ablaufes formal nachzuweisen. Ergebnis der MC ist entweder die Aussage, dass die zu prüfende Eigenschaft der Überprüfung standgehalten hat oder aber die Feststellung, dass dem nicht so war, wobei exemplarisch ein zur Verletzung führender Pfad ausgegeben wird. Da ein explizites Model Checking, bei dem alle Zustände des Zustandsraums in die Suche eingebunden werden, nur bei kleinen Systemen möglich ist, werden Alternativen verwendet, die sich in symbolische Verifikation (siehe unten), Zustandsraumerkundung (z.B. On-the-Fly MC) sowie Abstraktions- und Reduktionsmethoden aufteilen lassen. Zu letzteren gehören z.B. die Einteilung der Zustände in Äquivalenzklassen (Abstraktion), die Dämpfung der durch Nebenläufigkeit hervorgerufenen Zustandsraumexplosion (partielle Reduktion) sowie die Ausnutzung der natürlichen Symmetrie wiederholt aufgerufener Systemkomponenten (symmetrische Reduktion). Für weiterführende Informationen sei auf [Clarke et al., 1999, 2005] verwiesen.

Die im Rahmen dieser Arbeit eingesetzte Software verwendet die symbolische Modellverifikation. Hierbei wird das Systemverhalten auf boolesche Aussagen abgebildet und die Zustände durch Ja/Nein-Aussagen aufgegliedert. Die eigentliche Berechnung findet dann auf Formeldarstellungen der Pfade statt, welche z.B. mittels Binary Decision Diagrams (BDDs, [Lee, 1959; Akers, 1978; Bryant, 1986]) dargestellt werden können. Ein BDD ist eine kanonische Darstellung boolescher Funktionen $F : \{0,1\}^n \rightarrow \{false, true\}$ und topologisch ein direkter azyklischer Graph mit Form eines Entscheidungsbaums. Die verallgemeinerte (und für PMC verwendete) Form der BDDs sind die Multi-Terminal Binary Decision Diagrams (MTBDDs), welche von [Clarke et al., 1993] vorgeschlagen und von [Bahar et al., 1993; Clarke et al., 1997] weiterentwickelt wurden. Das größte Problem bei der Verwendung von (MT)BDDs ist, dass Größe, Struktur und somit auch die darauf basierende Verarbeitungsgeschwindigkeit derselben stark von der Reihenfolge der die Struktur aufspannenden Variablen abhängig sind. Näheres hierzu findet sich z.B. in [Beyer, 2002; Enders et al., 1991]. Wie [Bolling und Wegner, 1996; Tani et al., 1993] zeigten, ist die Lösung des Problems einer optimalen Struktur NP-hard, was für *at least as hard as any NP problem* [Garey und Johnson, 1979] steht, wobei NP (Non-deterministic Polynomial-time, [Cook, 1971]) bedeutet, dass sich dieses Problem (vermutlich) nicht mit polynomialem Zeitaufwand lösen lässt. Über die Jahre wurde eine ganze Reihe von Heuristiken zur Optimierung vorgeschlagen, von denen sich viele auf zwei Faustformeln abstrahieren lassen, nämlich, dass gekoppelte Variablen möglichst in der Nähe voneinander angeordnet sein sollten und dass stark gekoppelte Variablen früher im MTBDD verwendet werden sollten, als weniger stark gekoppelte (z.B. [Parker, 2002; Hermanns et al., 2003b]). Weitergehende Details zu MTBDDs und ihrer Verwendung zur wahrscheinlichkeitsbasierten Modellverifikation finden sich z.B. in [Kwiatkowska et al., 2004a].

Der größte Vorteil des Model Checkings liegt in der garantierten Berücksichtigung aller möglichen Evolutionspfade eines Systems (statt sich auf eine durch Simulation oder Testverfahren i.A. gegebene Untermenge zu beschränken). Als größter Nachteil erweist sich, dass der zu bildende Zustandsraum die unangenehme Eigenschaft besitzt, sehr stark mit der Komplexität des Modells anzuwachsen. Dieses, in lingua als Zustandsraumexplosion bekannte Problem, lässt sich bis zu einem gewissen Grade durch Anwendung bewährter Modellierungsregeln vermeiden. Daher wird bei der Beschreibung verteilter Systeme eine strikt modulare Modellierung und die Verwendung einer Vielzahl von Synchronisationstechniken empfohlen. Darüber hinaus sollte stets geprüft werden, ob die verwendeten Variablen den zugelassenen Wertumfang tatsächlich ausschöpfen und ob alle im Modell vorgesehenen Teilmodule und Verhaltensmodi auch wirklich benötigt werden.

Die wahrscheinlichkeitsbasierte Modellverifikation (PMC, Probabilistic Model Checking) ist eine Fortentwicklung der klassischen MC, beruht also auf den selben Prinzipien. Im Gegensatz zur MC liefert PMC jedoch keinen Fehlerpfad, sondern beantwortet lediglich die Frage, ob die Wahrscheinlichkeit dafür, dass die gewünschte Eigenschaft vom Systemmodell erfüllt wird, über oder unter einer beliebig wählbaren Schranke liegt. Da für diesen Größenvergleich jedoch zunächst der explizite Wahrscheinlichkeitswert ermittelt werden muss, unterstützen die meisten Implementierungen zusätzlich die Funktion, diesen Wert auch auszugeben. Dies macht PMC für die Anwendung als Methodik der Analyse von NAS sinnvoll anwendbar.

Die Modellwelt der PMC sind kontinuierliche und diskrete Markovketten (Continuous/Discrete Time Markov Chains, C/DTMC), sowie Markov-Entscheidungsprozesse (Markov Decision Processes, MDP). Eine Verwendung kontinuierlicher Markov-Ketten für die Modellierung von NAS scheitert an der Tatsache, dass diesen statt festen Schaltzeitpunkten exponentiell verteilte Übergangsraten zu Grunde liegen. Dies ist optimal, wenn man z.B. chemische Prozesse modellieren möchte. Für Netzbasierte Automatisierungssysteme hingegen ist die Beschränkung auf Exponentialverteilungen ein Ausschlusskriterium, da in diesen (und vielen anderen technischen Anlagen) feste Zeitintervalle oder Gleichverteilungen häufig auftreten, beispielsweise in Form von Zykluszeiten oder „time-outs“. Im Gegensatz zur Nutzung kontinuierlicher Markov-Ketten, eignet sich die diskrete Variante für die Modellierung von NAS, da deren Verwendung die Implementierung beliebiger Verteilungen ermöglicht – allerdings auch die Diskretisierung der Zeitachse erfordert (vgl. hierzu Abschnitt 5.5). MDPs schlussendlich sind erweiterte DTMCs, welche die Möglichkeit bieten, eine nicht-deterministische Übergangsauswahl zu modellieren. Diese Option wird für die Modellierung von NAS jedoch nicht benötigt.

Anmerkung: Im Rahmen dieser Arbeit wird zwischen deterministisch und nicht-deterministisch sowie stochastisch (i.S.v. wahrscheinlichkeitsbedingt) unterschieden. Während im deterministischen Fall immer nur eine Zustandsübergangsbedingung erfüllt sein darf, können es im nicht-deterministischen Fall mehrere sein. Im stochastischen Fall werden die Zustandsübergänge mit Eintrittswahrscheinlichkeiten gewichtet und somit eine Struktur

geschaffen, die genutzt werden kann, um ein bekanntes Verhalten zu abstrahieren. Da sich dies jedoch wiederum auf ein deterministisches Verhalten im Sinne des DTMCs abbilden lässt, wird im Weiteren zwischen deterministischem Verhalten mit und ohne stochastische Erweiterung auf der einen und nicht-deterministischem Verhalten auf der anderen Seite unterschieden.

Definition: Eine diskrete Markovkette (DTMC) ist ein Tupel $\mathbf{M} = (\mathbf{Z}, \mathbf{Z^0}, \mathbf{P})$, wobei $\mathbf{Z}$ für die Zustandsmenge, $\mathbf{Z^0} \subseteq \mathbf{Z}$ für die Menge der Anfangszustände und $\mathbf{P}$ für die Zustandsübergangsmatrix $\mathbf{P} : \mathbf{Z} \times \mathbf{Z} \rightarrow [\mathbf{0}, \mathbf{1}]$ steht. Letztere weist jedem Zustandsübergang einen Wahrscheinlichkeitswert zu. ■

Komplexe Strukturen werden durch die modulare Beschreibung eines DTMCs leichter darstellbar. Hierbei wird das DTMC $\mathbf{M}$ durch parallele Komposition mehrerer Teil-DTMCs $\mathbf{M_i}$ gebildet. Die zugehörigen Zustandsübergangsmatrizen $\mathbf{P}_i$ können dabei vom Verhalten des Restsystems abhängen, eine Abhängigkeit, die über die Menge $\mathbf{C}_i$ beschrieben wird: $\mathbf{P}_i : \mathbf{Z}_i \times \mathbf{C}_i \times \mathbf{Z}_i \rightarrow [0, 1]$. Näheres hierzu findet sich z.B. in [Parker, 2002].

Die Verwendung von Model Checking benötigt eine formale Spezifikation, deren Vokabular, Syntax und Semantik präzise definiert sind. Hierfür werden Kripke-Strukturen [Hughes und Cresswell, 1977] verwendet. Eine Kripke-Struktur entspricht einem labelled DTMC, d.h., jedem Zustand wird zusätzlich eine Menge atomarer Aussagen zugewiesen. Mittels Kripke-Strukturen lassen sich zwar Eigenschaften einzelner Zustände beschreiben, nicht jedoch dynamisches (Zeit-)Verhalten, wie dies z.B. für die Antwortzeitanalyse erforderlich ist. Daher wird zur Spezifikation der Eigenschaften eine formale, temporale Logik benötigt, die es ermöglicht, Eigenschaften zu definieren, die aus Elementaroperatoren oder logischen Verknüpfungen der Variablen im Modell bestehen und welche dann auf eine Kripke-Struktur abgebildet werden kann. Somit wird es möglich, dem Verifikationsalgorithmus sowohl ein auf endlichen Automaten basierendes Systemmodell als auch die zu prüfenden Eigenschaften in gleicher Darstellungsform zu übergeben.

Unter Zuhilfenahme temporaler Logiken lassen sich eine ganze Reihe von Fragestellungen angehen, wie beispielsweise Erreichbarkeit, Lebendigkeit oder Verklemmungsfreiheit [Bérard et al., 2001]. Die ersten für die formale Verifikation eingesetzten Logiken waren die Linear Temporal Logic (LTL, [Pnueli, 1977]) sowie die Computation Tree Logic (CTL, [Clarke und Emerson, 1981]). Diese boten jedoch noch keine Möglichkeit, Wahrscheinlichkeiten beinhaltende logische Konstrukte zu formulieren. Eine Notwendigkeit hierfür ergab sich auch erst beim Übergang zur PMC, was zu Logiken wie z.B PCTL und PTCTL führte. PTCTL entstand aus PCTL und TCTL [Alur et al., 1993], wobei letztere die Verwendung von „clock formulas“ (auch als *freeze variables* bekannt) ermöglicht. Hierdurch wird es möglich, das zeitliche Folgeverhalten betreffende Eigenschaften zu definieren, wie z.B. *auf A folgt B und dann innerhalb von 5 ms C, wobei A dazwischen erneut aktiviert worden ist.* Softwaremäßig wird PTCTL derzeit noch nicht ausreichend unterstützt, weshalb es im Rahmen dieser Arbeit keine Anwendung gefunden hat. Für weiterführende Informationen zu PTCTL wird auf [Laroussinie und Sproston, 2005; Kwiatkowska et al., 2007] verwiesen.

Das im Weiteren verwendete PCTL (Probabilistic Computation Tree Logic, [Hansson und Jonsson, 1994]) ist eine Erweiterung von CTL und kann zur Beschreibung von temporalen sowie stochastischen Systemen genutzt werden. PCTL führt den Operator $\mathrm{P}_{\sim p}\ [\ \cdot\ ]$ ein, mit $p \in [0,1]$ als Grenzwert der Wahrscheinlichkeit und $\sim \in \{\leq, \geq, <, >\}$. Beispielsweise kann mit $\mathrm{P}_{\geq 90\%}\ [\ x = 5\ ]$ geprüft werden, ob x mit mindestens 90%-iger Wahrscheinlichkeit (mindestens einmal) den Wert 5 annimmt. Da es oftmals nützlich ist, den genauen Wahrscheinlichkeitswert statt nur eines booleschen Ergebnisses zu erhalten, unterstützen viele Implementierungen auch einen P=? $[\ \cdot\ ]$ Operator (vgl. Abschnitt 3.1.2). Im Beispiel würde also mit P=? $[\ x = 5\ ]$ geprüft werden, mit welcher Wahrscheinlichkeit x (mindestens einmal) den Wert 5 annimmt. Näheres zur Syntax von PCTL findet sich z.B. in [Kwiatkowska et al., 2007].

Während es für die klassische Modellverifikation eine ganze Reihe von Softwarepaketen gibt (z.B. HyTech, UPPAAL, Kronos, SMV), ist die Auswahl für die wahrscheinlichkeitsbasierte Modellverifikation bislang noch sehr übersichtlich. Neben dem im Rahmen dieser Arbeit verwendeten PRISM (vgl. Abschnitt 3.1) sei an dieser Stelle lediglich auf MRMC [Katoen et al., 2005], VESTA [Sen et al., 2005b], YMER [Younes, 2005] sowie ProbVerus [Hartonas-Garmhausen et al., 1999] verwiesen[1]. Eine empirische Studie zum Vergleich einiger dieser Softwaretools findet sich in [Oldenkamp, 2007].

Hinweis: Ymer und VESTA verwenden die sogenannte statistische Modellverifikation. Das bedeutet, dass hier statt einer vollständigen Exploration des Systemverhaltens auf einen Monte-Carlo-Ansatz zurückgegriffen wird, bei dem die Berechnung erst gestoppt wird, wenn das zugehörige Ergebnis statistisch akzeptabel ist. Weitergehende Informationen finden sich in [Hérault et al., 2004; Sen et al., 2005a; Younes et al., 2006]. Die letztgenannten schlugen darüber hinausgehend vor, einen hybriden, aus klassischem und statistischem PMC bestehenden Ansatz zu verwenden, wobei das klassische PMC von den Autoren als numerisches PMC bezeichnet wird. Im Rahmen dieser Arbeit wird nur das numerische PMC verwendet.

3.1 Kurze Einführung in PRISM

In diesem Unterkapitel soll kurz auf die Arbeitsweise der im Rahmen dieser Arbeit verwendeten Modellverifikationssoftware PRISM eingegangen werden. PRISM [Kwiatkowska et al., 2002] wurde von der School of Computer Science der University of Birmingham entwickelt, wobei die Entwickler seit Juli 2007 an der University of Oxford arbeiten. PRISM unterstützt die Verwendung von DTMC-, CTMC- und MDP-Modellen. Die Modelle selbst werden hierbei in einer PRISM-eigenen Programmiersprache kodiert, welche in Abschnitt

[1] Nähere Informationen und Downloadmöglichkeiten der genannten Programme finden sich unter: http://www.eecs.berkeley.edu/~tah/HyTech, http://www.uppaal.com, http://www-verimag.imag.fr/TEMPORISE/kronos, http://www.cs.cmu.edu/~modelcheck/smv.html, http://www.tempastic.org/ymer, http://osl.cs.uiuc.edu/~ksen/vesta2, http://wwwhome.cs.utwente.nl/~zapreevis/mrmc und http://www.prismmodelchecker.org

3.1.1 kurz vorgestellt werden wird. Für die Spezifikation der Eigenschaften wird PCTL samt Erweiterungen zur Analyse von Pfadkosten sowie zur Ausgabe der Wahrscheinlichkeitswerte eingesetzt. Die Berechnung erfolgt unter Verwendung des in [Kwiatkowska et al., 2004b] vorgestellten hybriden Ansatzes. Dieser basiert darauf, dass MTBDDs in der reduzierten Form große Datenmengen (wie z.B. die Transitionsmatrix) sehr effizient repräsentieren können, während die Durchführung der für die Berechnung notwendigen Iterationen auf Basis von Sparse-Matrizen deutlich schneller ist. Hierdurch werden sowohl speicherplatz- als insbesondere deutliche geschwindigkeitsbasierte Verbesserungen erzielt.

3.1.1 PRISM-Programmiersprache

Die Modell-Eingabe erfolgt in einer PRISM eigenen Quellcodesprache, deren Struktur an dieser Stelle in aller Kürze vorgestellt werden soll. Für eine detailliertere Einführung wird auf [Parker et al., 2005] verwiesen. Wie Abbildung 5 zeigt, besteht der PRISM-Modell-Quellcode aus fünf Strukturelementen:

- Dem Schlüsselwort für den Modelltyp. Im Falle von DTMC ist dies `probabilistic`.
- Der Auflistung im Programm verwendeter Konstanten. Jeder Konstante ist dabei das Schlüsselwort `const` (und im Falle von reellwertigen Konstanten zusätzlich `double`) voranzustellen. Wird einer Konstanten im Code kein Wert zugewiesen, handelt es sich um einen freien Modellparameter, der erst zur Ausführung der Verifikation festgelegt werden muss.
- Der Auflistung von mit `formula` eingeleiteten textuellen Ersetzungsbefehlen (Abkürzungen). Derartige Elemente sind insbesondere zur Reduktion des Schreibaufwandes – und damit indirekt zur Steigerung der Lesbarkeit – interessant.
- Die Moduldefinitionsblöcke. Jeder Block beginnt mit dem Schlüsselwort `module` und endet mit `endmodule`.
- Ertragsfunktionen (`reward`). Hiermit können ggf. Kosten über Pfade definiert werden.

```
probabilistic                                          \\ Modelltyp
const double p=0.3;                                    \\ Konstantendefinition
const Dt;
formula xn=func(min,s,Dt);                             \\ Abkürzungen
module Grundstruktur                                   \\ Modulbeginn
  s :    [0..5]  init 0;                               \\ Variablendefinition
  [pre]  s=0     -> s'=xn+1;                           \\ Befehlszeile mit pre-Sync
  [t]    s>0     -> p:(s'=5) + (1-p):(s'=xn);          \\ Befehlszeile mit t-Sync
endmodule                                              \\ Modulende
```

Abbildung 5: Grundstruktur des PRISM-Modellspezifikationscodes (ohne Reward-block)

Ein Moduldefinitionsblock besteht stets aus einem Deklarations- und einem Anweisungsteil:

- Variablendeklarationen: Diese bestehen aus dem Namen der Variablen, dem Wertebereich und ggf. dem zu verwendenden Anfangswert.
- Anweisungszeilen: Diese bestehen aus einem Synchronisationsmarker, einer Bedingung, dem Zuweisungspfeil sowie der eigentlichen Zuweisung. Der Synchronisationsmarker besteht aus zwei rechteckigen Klammern, die ggf. eine Synchronisationskennung umschließen. Ist eine solche Kennung vorhanden, bedeutet dies, dass alle im Programm enthaltenden Modulblöcke, die derart gekennzeichnete Zeile(n) aufweisen, eine solche Befehlszeile nur ausführen dürfen, wenn alle anderen Module ebenfalls eine solch markierte Zeile ausführen. Die Übergangsbedingung besteht aus mittels UND oder ODER verknüpften Zustandsbedingungen (z.B. `x=3`), wobei diese auch negiert sein können. Die Zuweisung muss die folgende Struktur aufweisen:

  ```
  p1:(x1'=3)&(x2'=5)+p2:(x1'=2)&(x2'=x1+x2)
  ```

  Hierbei sind `x1` sowie `x2` Variablen und `p1` sowie `p2` Wahrscheinlichkeiten. Die Summe der Wahrscheinlichkeiten einer Zuweisung muss stets zu eins addieren. Der hochgestellte Strich nach den Variablennamen zeigt an, dass es sich um den im nächsten Schritt aktiven Wert der Variablen handelt. Rechts des Gleichheitszeichens können Konstanten oder Funktionen stehen. Die einzelnen mit Eintrittswahrscheinlichkeiten `pi` gewichteten Zuordnungen werden durch ein `+` getrennt. Folgen auf eine Wahrscheinlichkeit mehrere Variablenzuweisungen, sind diese durch ein `&` zu verknüpfen. Gibt es keine Wahrscheinlichkeitsaufsplittung (d.h. `p1=1`), kann `p1` – inklusive des Doppelpunkts – weggelassen werden.

Am Ende jeder einzelnen Zuweisung muss ein Semikolon stehen. Die Definition von Arrays ist leider ebenso wenig möglich, wie die Verwendung rekursiver Strukturen (auch wenn letztere in DTMC durchaus möglich wären).

3.1.2 Formulierung der Eigenschaften in PCTL

Im Rahmen dieser Arbeit wurden die beiden PCTL-Operatoren

`P=? [ expr1 U expr2 ]`	(stochastisches `Until`)
`P=? [ expr1 U<=k expr2 ]`	(beschränktes (bounded) `Until`)

sowie der Rewardbefehl `R=? [F expr1]`

verwendet. `expr1` und `expr2` sind logische Aussagen, welche boolesche Ergebnisse haben. Während sich mit den ersten beiden Operatoren (`P=?`) Wahrscheinlichkeiten ermitteln lassen, dient der angegebene Rewardbefehl (`R=?`) dafür, die mittleren Kosten zu bestimmen, um zu einem Zustand zu gelangen, der die Bedingung `expr1` erfüllt. Für letzteren muss vorab jedem Zustandsübergang ein (i.d.R. reellwertiger) Wert zugewiesen werden, der ein Maß für die durch diesen Wechsel repräsentierten Kosten (bzw. Erträge) darstellt. Ist

dieser Wert für alle Zustandsübergänge gleich groß, ließe sich diese, mittels (`R=?`) erhaltene Information auch aus der Kenntnis einer Verteilungsfunktion der Schrittanzahl (wie sie mit einem der obigen (`P=?`) Operatoren ermittelbar ist), extrahieren; allerdings dauert dies deutlich länger. Sind die Kosten der einzelnen Zustandsübergänge nicht identisch, so kann mit Hilfe des Rewardbefehls relativ elegant eine Information gewonnen werden, die ansonsten lediglich unter Zuhilfenahme zusätzlicher Modellvariablen erzeugbar wäre. Für den praktischen Einsatz von PMC ist die Option, unter Verwendung des Rewardbefehls schnell mittlere Laufzeiten ermitteln zu können, also durchaus interessant.

Die beiden Wahrscheinlichkeiten ermittelnden Operatoren lesen sich wie folgt: Bestimme die Wahrscheinlichkeit, dass `expr1` (mindestens) so lange eine wahre Aussage darstellt, bis `expr2` erfüllt ist. Damit ist lediglich gefordert, dass `expr2` irgendwann erfüllt sein muss, nicht jedoch, dass `expr2` auch erfüllt bleibt – `expr2` könnte also anschließend auch wieder eine falsche Aussage darstellen oder gar erneut einen erfüllten Wert annehmen, ein Verhalten, das mit diesem Operator leider nicht erfasst wird! Der Unterschied zwischen dem oberen und dem unteren der beiden (`P=?`) Operatoren besteht darin, dass das bounded Until (`U<=k`) die zu untersuchende Zeitachse auf die ersten `k`-Schritte beschränkt, während das stochastische Until (`U`) die komplette Systemevolution (ohne Zeitschranke) berücksichtigt.

Die einfachste Weise, diese beiden Operatoren zu verwenden, besteht darin, `expr1` durch true zu ersetzen und lediglich mit der zweiten Aussage (`expr2`) zu arbeiten (`P=? [ true U expr2 ]` bzw. `P=? [ true U<=k expr2 ]`). Somit soll nur noch die Wahrscheinlichkeit dafür bestimmt werden, dass `expr2` irgendwann (bzw. innerhalb von k Schritten) erfüllt worden sein wird. Ist z.B. die Zeitverteilung einer bestimmten Aktion gesucht, so kann dies entweder indirekt mittels der schrittbeschränkten Version `P=? [ true U<=k expr2 ]` oder direkt, unter Einführung einer Zählervariablen, mittels `P=? [ true U Runtime=Lf ]` ermittelt werden. `Lf` ist ein Parameter, wobei die Eigenschaft für jeden innerhalb eines vorgegebenen Intervalls liegenden Wert von `Lf` überprüft wird. `Runtime` ist eine Variable, welche durch das Modell gesetzt wird, indem, sobald die überwachte Aktion eingetreten ist, der aktuelle Wert eines Laufzeitzählers in die Variable `Runtime` geschrieben wird. Der Vorteil dieser Vorgehensweise liegt darin, dass man direkt die einzelnen (diskreten) Wahrscheinlichkeitswerte erhält, während man bei Test auf den Zählerstand des ursprünglichen Laufzeitzählers den Wert der Verteilungsfunktion (ausgewertet von `Lf` bis unendlich) zurückgeliefert bekommt. Soll hingegen die Wahrscheinlichkeit ermittelt werden, dass ein Ereignis innerhalb einer gesetzten Zeitspanne eintritt, ist der (von beiden `Until`-Operatoren direkt – oder als Differenz zu eins – ermittelte) Integralwert die gesuchte Größe.

Beim stochastischen `Until` wird also zusätzlich eine (globale) Uhrvariable benötigt, welche die zu ermittelnde Zeitdifferenz explizit misst und im Anschluss lediglich ausgelesen werden muss. Beim beschränkten `Until` ist zwar keine weitere Uhrvariable erforderlich, dafür muss aber stets der komplette Zustandsgraph ausgewertet werden. Dies führt dazu, dass die schrittbeschränkte Auswertung (`U<=k`) in einem gegebenen System immer ungefähr die gleichen Anforderungen an Speicherplatz und Rechenzeit stellt, unabhängig von dem tatsächlichen Abstand der beiden Ereignisse. Dahingegen weist die unbeschränkte

Auswertung (U) ein exponentielles Bedarfsverhalten auf, welches insbesondere dem zusätzlichen Speicherplatz geschuldet ist (vgl. hierzu auch Abbildung 6). Bei gleicher Modellgröße benötigt die Verwendung des beschränkten Operators meistens etwas mehr Zeit als die des unbeschränkten und ist vom rein analytischen Aufwand her in jedem Fall teurer, i.S.v. aufwändiger (vgl. [Hermanns et al., 2003a]). Einen großen Vorteil erzielt die Verwendung der beschränkten Version dann, wenn hierdurch auf die direkte Implementierung einer Gesamtzeit verzichtet werden kann. Dies ist insbesondere dann der Fall, wenn a) mit einer konstanten Zeitschrittweite gearbeitet wird und b) das Modell keine Absolutzeit benötigt, diese Variable also nicht explizit im Modellcode auftauchen muss. Wichtig zu klären ist daher, ob eine Uhr benötigt wird, die die Gesamtzeit misst oder ob alle Funktionen mittels lokaler Uhren abgedeckt werden können. Der Verzicht auf eine solche absolute Uhr führt zu einer deutlichen Reduktion der Modellgröße und somit auch des Ressourcenbedarfs, wie Abbildung 6 exemplarisch zeigt.

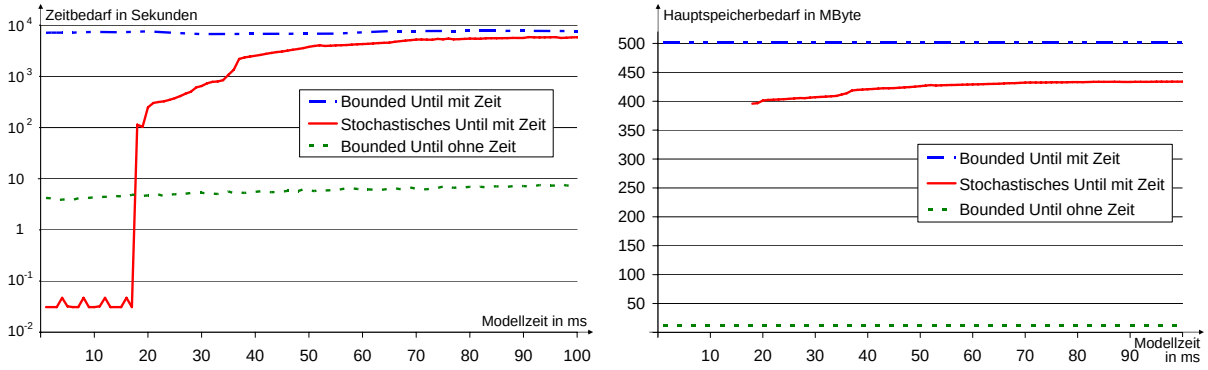

Abbildung 6: Zeit- (links) und Speicherbedarf (rechts) für das in Abschnitt 6.3 vorzustellende Beispiel eines mit Zugriffskonflikten sowie Netz- und Sensorstörungen beaufschlagten NAS. Die einzelnen Trajektorien repräsentieren die Verwendung des stochastischen Until-Operators für ein Modell mit absoluter Zeitvariablen und des beschränkten Until-Operators für je eine Modellvariante mit und eine ohne absoluter Zeitvariablen. Die Berechnung erfolgte auf einem Intel Pentium4 3 GHz Prozessor mit 1 GByte Hauptspeicher.

Nicht verschwiegen werden soll, dass ab einer gewissen Modellgröße der verfügbare Hauptspeicher die geschwindigkeitsbestimmende Größe ist: Solange das komplette Modell im Hauptspeicher Platz findet, ist PMC um ungefähr einen Faktor 100 schneller, als wenn Teile davon ausgelagert oder zwischenzeitlich (zur Speicherbereinigung) gelöscht werden müssen. Der benötigte Hauptspeicher ist dabei stark von der Größe des der PMC-Analyse zugrundeliegenden MTBDDs abhängig. Als grobe Faustformel kann von 23 Byte pro Knoten ausgegangen werden, wobei die Komplexität von PCTL Model Checking linear bzgl. der Formelgröße und polynomial im Zustandsraum [Kwiatkowska, 2003] ist. Neben dem zeitlichen Aspekt, wird die mögliche Maximalgröße und Komplexität der Modelle auch durch das von PRISM verwendete CUDD-package (Colorado University Decision Diagram package [Somenzi, 1997]) beschränkt, da CUDD nicht dafür geschrieben wurde, sehr große Zustandsräume zu verarbeiten (maxmem $\approx$ 1,85 GByte).

3.2 PMC-Ablauf

Wie bereits erläutert, wird bei der Modellverifikation zunächst ein Modell des Systems sowie eine formalisierte Fassung der zu überprüfenden Eigenschaften erstellt. Diese formalen Beschreibungen von Modell und Eigenschaften finden dann Eingang in den Verifikations-Algorithmus, welcher prüft, ob das Systemmodell die geforderten Eigenschaften erfüllt (Abbildung 7). Hierfür sind zwei Arbeitschritte notwendig, namentlich Modellerstellung und Testen. Im ersten Schritt wird ein, das übergebene Systemmodell repräsentierender, MTBDD gebildet. Dieser Arbeitsschritt ist somit unabhängig von den übergebenen, zu verifizierenden Eigenschaften, welche erst im zweiten Schritt Verwendung finden. Dieser zweite Schritt beinhaltet die Erzeugung weiterer MTBDDs oder Sparse-Matrizen sowie diverse iterative Berechnungen. Die erste Phase muss daher nur ein einziges Mal durchlaufen werden, unabhängig davon, wieviele Eigenschaften auf diesem Systemmodell geprüft werden sollen. Somit kann es durchaus interessant sein, Modellkomplexität – welche sich auf die Modellerstellungszeit und die Testphase auswirkt – und Eigenschaftskomplexität – welche lediglich Einfluss auf die Dauer der Testphase hat – gegeneinander abzuwägen. Dies gilt natürlich nur, sofern die Eigenschaften nicht in Abhängigkeit systemverändernder Parameter formuliert worden sind.

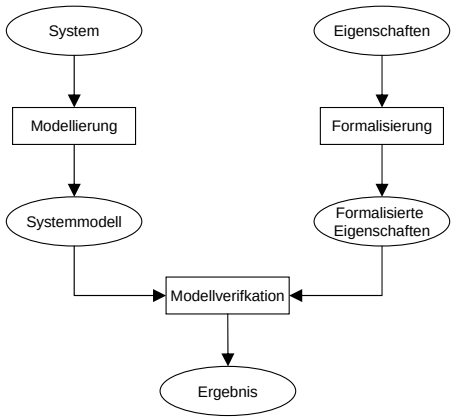

Abbildung 7: Workflow der klassischen Modellverifikation (MC)

3.2.1 Entwurfsprozess

Im Gegensatz zur klassischen Modellverifikation (MC), bei der System und Eigenschaften unabhängig voneinander formalisiert werden können (d.h. eine Änderung des Systems beeinflusst lediglich das Systemmodell, nicht jedoch die formale Eigenschaftsbeschreibung und umgekehrt), ist dies für die wahrscheinlichkeitsbasierte Modellverifikation (PMC) nicht der Fall. Neben der allgegenwärtigen Zustandsraumexplosion ist hierfür die Notwendigkeit zur Terminierung (Abschnitt 3.2.2), zur Transformation der Systemzeit auf den Ereigniseintritt (Abschnitt 3.2.3) und zur Diskretisierung der stochastischen Modelle (Abschnitt 5.5) verantwortlich. In der Folge entsteht ein optimiertes Systemmodell, das auf die zu prüfende Eigenschaft zugeschnitten ist und sich auch nur zur Verifikation dieser (sowie

direkt „benachbarter“ Eigenschaften) verwenden lässt. Dementsprechend verändert sich der Entwurfsprozess insofern, als dass die Trennung zwischen Modell und Eigenschaften aufgegeben und ein neuer formaler Entwurfsprozess gefunden werden muss (Abbildung 8).

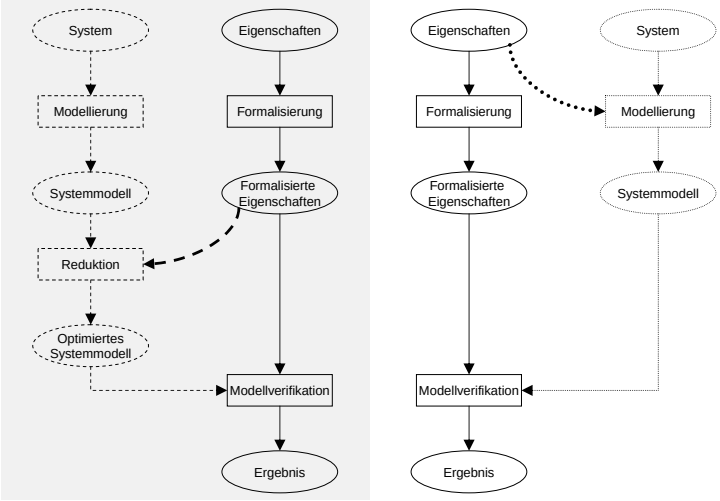

Abbildung 8: Workflow der wahrscheinlichkeitsbasierten Modellverifikation (PMC)

Während die Eigenschaften wie zuvor formuliert werden, muss der Entwurfsprozess des Modells angepasst werden. Die erste Möglichkeit dies zu tun ist in gestrichelten Linien auf der linken Seite von Abbildung 8 dargestellt: Das generische Systemmodell wird mit den spezifischen formalisierten Eigenschaften zusammengefügt. Dieses Vorgehen erfordert eine anschließende Reduktion des entstandenen Modells und das nachträgliche Einfügen der Abbruchbedingung(en). Der zweite mögliche Weg, in Abbildung 8 in gepunkteter Linie auf der rechten Seite dargestellt, besteht darin, die Eigenschaftsbeschreibungen bereits in den Modellierungsschritt mit einzubeziehen. Dies führt zu sehr viel eleganteren Modellen, erfordert jedoch vom Anwender, das Modell für jede zu prüfende Eigenschaft neu zu erstellen. Im Rahmen dieser Arbeit wurde der zweite Ansatz verwendet, da es zunächst erforderlich war, ein sehr genaues Verständnis dessen zu erlangen, wie ein als optimal anzusehendes Modell aufgebaut sein muss. Für zukünftige Arbeiten wäre es jedoch durchaus denkbar, die vorhandene Erfahrung im Umgang mit entsprechenden Modulen in einen sinnvollen Reduktionsansatz umzusetzen und somit den im gestrichelten, linken Ast beschriebenen Ansatz wählen zu können.

3.2.2 Terminierungsnotwendigkeit

Typische Fragestellungen im Zusammenhang mit PMC beziehen sich auf

(a) die Wahrscheinlichkeit, dass innerhalb einer bestimmten Zeit eine Veränderung aufgetreten ist (z.B. ein Fehlerfall),

(b) die Zeitverteilung für das Erreichen eines eingeschwungenen Zustands,

(c) die relativen Anteile einer Zielmenge oder

(d) die Verteilung der Zeitdauer bis zum Erreichen einer Zielmenge.

Alle diese Fragestellungen haben eines gemeinsam, nämlich einen wohldefinierten Schlusspunkt. Dies jedoch wird zum Problem, wenn das System zyklische oder andere sich wiederholende Prozesse aufweist, und somit eine zeitliche und ablaufbasierte Zuordnung von Anfangs- und Zielmenge vorgenommen werden muss. Eine solche Situation ist beispielsweise gegeben, wenn das Antwortzeitverhalten eines NAS untersucht wird. Hierbei muss sichergestellt sein, dass eine Reaktion am Ausgang tatsächlich von dem betrachteten Eingangsimpuls und nicht von einem verspätet ankommenden früheren Eingangsimpuls ausgelöst worden ist. In anderen Worten muss also dafür Sorge getragen werden, dass lediglich ein einziger Systemdurchlauf in der Analyse berücksichtigt wird, d.h. die Verifikation beendet wird, sobald der betrachtete Prozess einmal abgelaufen ist.

Hierzu ein Beispiel: Angenommen, es sei zu untersuchen, ob (bzw. mit welcher Wahrscheinlichkeit) im System eine bestimmte Eigenschaft (z.B. ein Fehlverhalten oder eine gegebene Übertragungszeit) in der Folge eines wiederholt auftretenden Signalwechsels anzutreffen ist. Vereinfacht lässt sich dies durch das in Abbildung 9 gezeigte, drei Zustände umfassende, Modell zeigen.

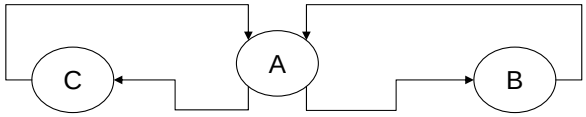

Abbildung 9: Vermeidung geometrischer Reihenentwicklung

Der mittlere Zustand (A) repräsentiert die Menge der Anfangszustände, also jenes Systemverhalten, das vorliegt, solange kein Eingangssignalwechsel aufgetreten ist. Zustand B und C stehen für das infolge des Eingangssignalwechsels ausgelöste Folgeverhalten, wobei Zustand B für das die Eigenschaft nicht erfüllende und Zustand C für das die Eigenschaft erfüllende Verhalten stehen möge. Schränkt man die Betrachtung nun nicht auf einen einzigen repräsentativen Signalwechsel und dessen mögliches Folgeverhalten ein, so ließe sich die Wahrscheinlichkeit, irgendwann einen die vorgegebene Eigenschaft erfüllenden Zustand zu erreichen, als geometrische, zu eins konvergierende Reihe beschreiben. Umgekehrt formuliert bedeutet dies, dass die (gesuchte) Wahrscheinlichkeit für den Eintritt in den die Eigenschaft erfüllenden Zustand C nur ermittelt werden kann, wenn die Verifikation beendet wird, bevor der Automat wieder nach A zurückkehrt (von wo aus er erneut nach B oder C wechseln wird).

Während sich dieses Problem in diesem einfachen Beispiel noch mittels `P=?[!B&!C U C]` umgehen lassen würde, ohne das Systemmodell anpassen zu müssen, ist dies für komplexere Beispiele nicht oder nur eingeschränkt möglich. Aus diesem Grunde muss die erste an ein mit Übergangswahrscheinlichkeiten formuliertes Modell zu stellende Bedingung lauten, dass die Abarbeitung stoppt, sobald das zu überwachende Ereignis eingetreten ist, bzw. ein Systemdurchlauf abgeschlossen wurde. Daraus folgt, dass (um das Problem vollständig abdecken zu können) einerseits jede mögliche Zustandskombination in diesem ersten Durchlauf erreicht werden muss, und das Modell andererseits selbstständig feststellen muss, dass das gesuchte Ereignis eingetreten ist (vgl. Abschnitt 4).

Die Notwendigkeit, PMC so intelligent gestalten zu müssen, dass es „selbstständig erkennt“ dass ein Systemdurchlauf abgeschlossen ist, beinhaltet auch die Chance, die Modellevolution sofort nach Eintritt des Ereignisses zu stoppen, und somit die für PMC benötigten Ressourcen (Speicher, Rechenzeit) zu verringern. Dies illustriert Tabelle 2.

Tabelle 2: PMC-Ressourcenverbrauch mit und ohne Terminierung für das in Kapitel 6.3 vorzustellende (nicht fehlerbehaftete) Zugriffskonfliktbeispiel unter Verwendung des stochastischen `Until`-Operators. Die Berechnung wurde auf einem Rechner mit Intel Pentium4 3 GHz Prozessor und 1 GByte Hauptspeicher ausgeführt.

	Modellerstellung		Eigenschaftsprüfung	
	Zustände	Zeitbedarf	Speicherbedarf	Zeitbedarf
ohne Terminierung	61.782.541	40,3 s	1.209.875 kB	520,3 s
mit Terminierung	5.135.181	24,8 s	105.315 kB	46,3 s

Hierbei wurde für das in Kapitel 6.3 vorzustellende Zugriffskonfliktbeispiel dargestellt, welche Ressourcen einerseits für die Modellerstellung und andererseits für eine spezielle Eigenschaftsprüfung (Wahrscheinlichkeit dafür, dass die Antwortzeit im 35. Zeitintervall liegt) benötigt werden, wenn PMC nach einem Durchlauf beendet wird sowie, wenn die Systemevolution fortgesetzt wird (wobei natürlich die für die Eigenschaftsprüfung erforderliche Bedingung ausgekoppelt wurde, so dass in beiden Fällen das korrekte Ergebnis erzielt werden konnte). Deutlich zu erkennen ist, dass die Terminierung den Ressourcenverbrauch in diesem typischen Anwendungsbeispiel um über 90% senken kann.

Eine weitere Möglichkeit, den Ressourcenverbrauch zu vermindern, besteht darin, die zeitliche Dynamik des zu erzeugenden Systemmodells noch weiter einzuschränken: Beim bisherigen Ansatz werden alle zum Zielbereich führenden Pfade ausgeführt, unabhängig davon, wie lange der zugehörige Pfad eigentlich ist. Insbesondere für komplexe, viele stochastischen Verzweigungen enthaltende Systeme ist dies jedoch meist nicht notwendig. Ein typisches Beispiel hierfür wäre die Situation, dass man eigentlich nur an jenen Pfaden interessiert ist, die ihr Ziel innerhalb einer gewissen Zeit erreichen. In diesem Fall kann man die Systemevolution nach dieser Zeit problemlos abbrechen. Abbildung 10 zeigt dies exemplarisch für das bereits für Tabelle 2 verwendete Beispiel. Die auf der Abszisse abgetragene Größe „Maxtime in ms“ zeigt an, nach dem wievielten Systemzeitschritt (à 1 ms) das System unabhängig von seinem aktuellen Zustand terminiert wird. Die unterste der Kurven dokumentiert die Zeit, welche zur Modellerzeugung notwendig ist. Diese lässt sich in drei Bereiche unterteilen: Während der erste und letzte Bereich dadurch gekennzeichnet ist, dass die gesuchte Wahrscheinlichkeit 0 beträgt, weist sie für den mittleren Bereich Werte zwischen 0 und 1 auf. In der Abbildung ist dies durch (in erster Näherung) logistisches Anwachsen des Funktionswertes erkennbar.

Die oberste Kurve zeigt den für die Eigenschaftsprüfung notwendigen Hauptspeicher, der bis 54 ms ungefähr linear ansteigt. Größere Werte von Maxtime haben – aufgrund der

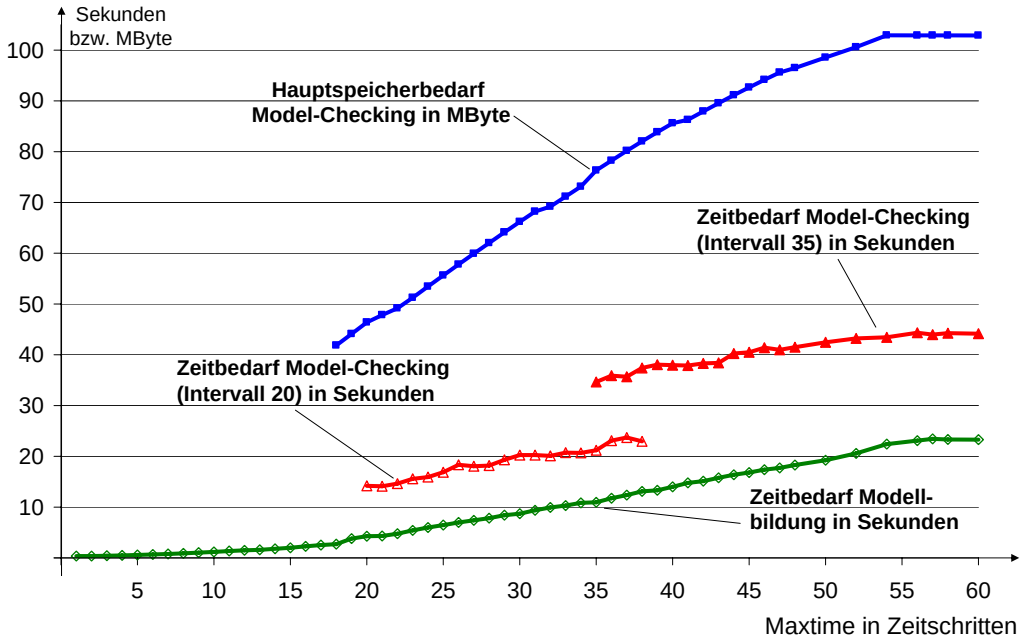

Abbildung 10: PMC-Ressourcenverbrauch bei reduzierter Systemevolution (stoch. `Until`)

Terminierung der Berechnungen – keinen weiteren Einfluss auf den Hauptspeicherbedarf der Modellprüfung. Zwischen 54 und 57 gibt es allerdings noch einen geringen Anstieg, der aber aufgrund des auf diesen Zeitbereich entfallenden Anteils von gerade 2,2‰ aller Evolutionen vernachlässigbar klein ausfällt (Vorsicht: dies gilt für den Hauptspeicherbedarf, nicht jedoch für die Rechenzeit!).

Die verbleibenden beiden Kurven repräsentieren die für die Eigenschaftsprüfung notwendige Zeit, wobei die linke der beiden Kurven eine Eigenschaftsprüfung für das bei 20 ms liegende Zeitinvall repräsentiert, während die rechte stattdessen 35 ms verwendet. Logischerweise kann die 35 ms Kurve erst bei Maxtime=35 beginnen, während die 20 ms Kurve bereits bei 20 ms beginnen konnte. Deutlich zu erkennen ist, dass der hier verwendete stochastische `Until`-Operator für bei größeren Zahlenwerten angesiedelten Zeitintervallen mehr Zeit benötigt als für niedrigere (weshalb der 35 ms Graph bei einem größeren Funktionswert beginnt, als ihn der 20 ms Graph für eine Maxtime von 35 ms aufweist). Der Anstieg der Kurven kann in beiden Fällen als ungefähr linear abstrahiert werden, was jedoch (insbesondere im Einzelfall) nicht immer zutrifft. Grund hierfür sind einerseits Verzögerungen durch im Hintergrund laufende Systemprozesse und andererseits die Nichtlinearität des PMC-Algorithmus.

3.2.3 Ermittlung des Anfangszustands

Bisher wurde lediglich beschrieben, dass die von einem externen Signal ausgelöste Systemevolution aus Gründen der Terminierung zu beobachten ist, nicht aber, wie der diese Beobachtung auslösende Signalwechsel modelliert wird. Klassischer Weise wird hierbei das

reale Systemverhalten nachgestellt, d.h., das System agiert in der gewohnten Weise, bis zu einem zufälligen Zeitpunkt ein externes Eingangssignal auftritt. Abbildung 11a zeigt dies beispielhaft für einen aus sieben sequentiell angesprungenen Zuständen bestehenden Prozess. Die Zustandsverweilzeiten wurden hierfür zu je einer Zeiteinheit für die Zustände zwei, drei, fünf und sieben sowie zwei Zeiteinheiten für die übrigen Zustände, angenommen. Da das externe Eingangssignal zu einem beliebigen, nicht näher bestimmten Zeitpunkt auftreten soll, wird die Wahrscheinlichkeit dafür, dass es zu einem bestimmten Zeitpunkt auftritt, als gleichverteilt angenommen (Abbildung 11b).

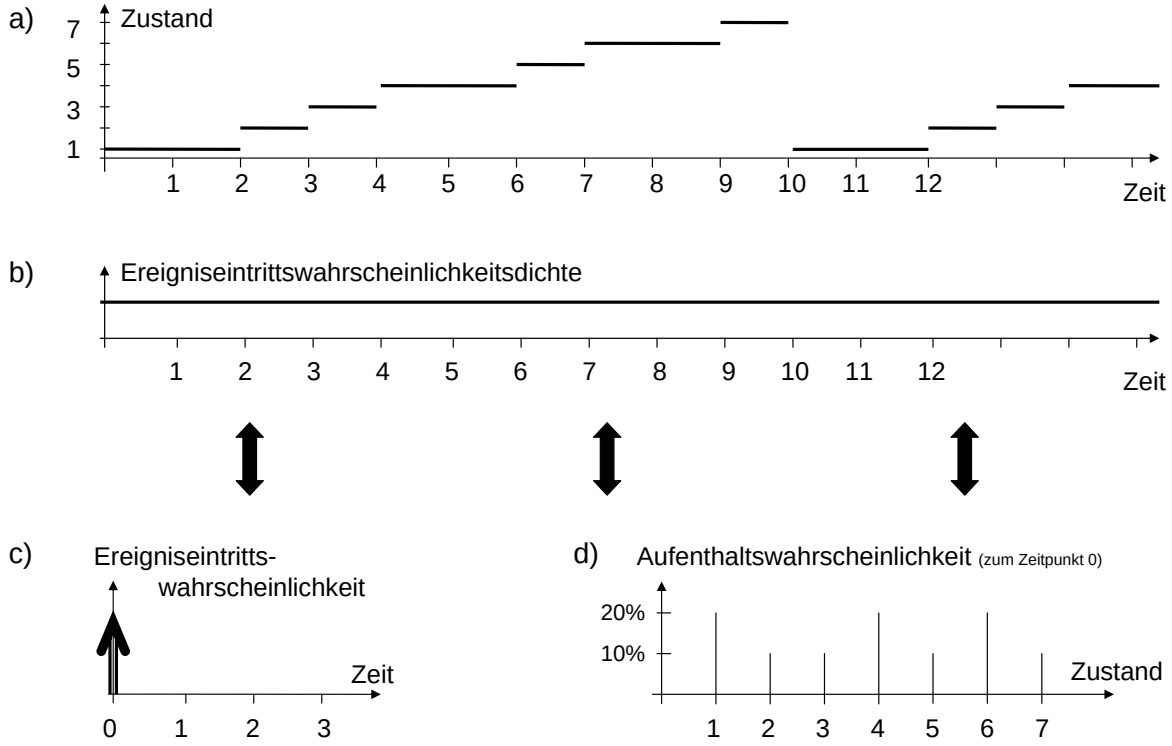

Abbildung 11: Transformation eines, zum Zeitpunkt 0 gestarteten und dann auf den Eintritt eines zufälligen Ereignisses wartenden, zyklischen Prozesses auf eine ereigniseintrittszeitpunktsbezogene Achse.

Die eigentliche Schwierigkeit liegt nun in der Implementierung des Signalgenerators, da die vorgegebene Auftrittswahrscheinlichkeit auch im – durch die Zustandsexplosion – zeitlich eng begrenzten Intervall vorhanden sein muss. In der Regel ist es hierbei erforderlich, eine zusätzliche Uhr im System zu integrieren, über deren Wertebereich der Ereigniseintritt erfolgt. Dies jedoch führt zu einer gravierenden Vergrößerung des Zustandsraums und somit der Berechnungsdauer. Darüber hinaus stellt sich das bereits von simulativen und messtechnischen Ansätzen her bekannte Problem, dass die einzelnen Teilsysteme sich mit der Zeit so gegeneinander verschieben müssen, dass tatsächlich alle möglichen im realen System in beliebiger Zeit auftretenden Kombinationsmöglichkeiten im PMC-Zeitintervall mittels ihrer tatsächlichen Wahrscheinlichkeit vorhanden sind.

Im Rahmen dieser Arbeit wurde daher ein weiterer Ansatz entwickelt. Dahinter steht die Überlegung, dass es in Bezug auf die zuvor geforderte Terminierung am einfachsten wäre, wenn das Gesamtsystemverhalten so transformiert werden könnte, dass alle zu betrachtenden Systemevolutionen vom gleichen Punkt aus starten. Das Systemmodell muss daher so aufgebaut sein, dass es sich zum (neuen) Zeitpunkt „Null“ so präsentiert, wie ein externer Beobachter es zu einem beliebigen Zeitpunkt der ursprünglichen Zeitachse antreffen würde. In anderen Worten bedeutet dies, dass die PMC-Implementierung nicht mehr den Ereigniseintrittszeitpunkt als zufällige Größe behandeln wird. Stattdessen wird der Systemzustand auf diese – durch den Ereigniseintritt gegebene – zufällige Zeitachse projiziert, was bedeutet, dass statt dem Ereigniseintrittszeitpunkt der Systemzustand als zufällige Größe behandelt wird: Nach erfolgter Transformation befindet sich der Ereigniseintrittszeitpunkt zwar zur Zeit 0 (Abbildung 11c), dafür ist jedoch der Systemzustand zu diesem Zeitpunkt nicht mehr eindeutig bestimmt, sondern ergibt sich als Aufenthaltswahrscheinlichkeitsvektor (Abbildung 11d). Jedem Zustand wird hierbei die Wahrscheinlichkeit dafür zugewiesen, dass das System vor der Transformation bei Ereigniseintritt in diesem Zustand gewesen wäre. Im Falle obigen Beispiels kann diese Information direkt als Quotient aus der durchschnittlichen Zeit, welche das System in einem Zustand verweilt und der durchschnittlichen Gesamtzeit (10 Zeiteinheiten), die das Sytem für einen Durchlauf benötigt, ermittelt werden. Es ergeben sich also 20% für die Zustände eins, vier und sechs sowie 10% für die übrigen Zustände. Für den generalisierten Fall einer allgemeinen Ereigniseintrittsdichtefunktion muss obiger Quotient noch mit dem Integral dieser Dichtefunktion über den zugehörigen Zeitabschnitten multipliziert werden.

Die Forderung nach einem einzigen, aber parallelisierten Durchlauf zwingt den Ingenieur dazu, den Entwurfsprozess (bzw. das System in selbigem) grundlegend neu zu durchdenken, weshalb sich Kapitel 4 ausführlich mit dem Problem der Anfangswertbestimmung auseinander setzen wird.

3.3 Mathematische Grundlagen

Zum Abschluss dieses Kapitels sollen noch kurz die wichtigsten, PMC zugrunde liegenden, mathematischen Zusammenhänge dargestellt werden. Diese wurden aus den Arbeiten von Kwiatkowska et al. zusammengestellt.

3.3.1 Pfadwahrscheinlichkeiten und Kosten in DTMC

Die bei der Ausführung eines mittels DTMC modellierten Systems entstehende nicht-leere Zustandsfolge $\mathbf{z}_0$, $\mathbf{z}_1$, ... $\mathbf{z}_l$, mit $\mathbf{z}_k \in \mathbf{Z}$ wird als **Pfad** ω bezeichnet. $\mathbf{Z}$ stellt dabei die Menge aller Zustände der DTMC dar, $\mathbf{z}_0$ den Anfangszustand des Pfades und $|\omega| = l$ die als Pfadlänge l bezeichnete Anzahl an Zustandsübergängen in der durch ω gegebenen Sequenz. Der zeitliche Abstand zweier in einem Pfad ω benachbarten Elemente $\mathbf{z}_i$, $\mathbf{z}_{i+1}$, also die Zeit, welche vom Verlassen des i-ten Zustands bis zum Verlassen des (i+1)-ten Zustands vergeht, wird mit Δt bezeichnet und im Rahmen dieser Arbeit als von i unabhängig angenommen.

Im Umkehrschluss bedeutet dies, dass sich die Zeitverzögerung zwischen dem Verlassen (oder Erreichen) zweier Zustände aus der Länge des Pfades von einem zum anderen Zustand, multipliziert mit Δt ergibt. Folglich ist der Zeitpunkt eines Zustandes mit seiner Position in einer vom Initialzustand zu ihm führenden Kette von Zuständen identisch.

Die Menge aller, einen Zustand $\mathbf{z}_i$ passierenden Pfade wird mit $Paths(\mathbf{z}_i)$ bezeichnet. Die Menge aller Pfade $Paths(\mathbf{z}_j|\mathbf{z}_i)$, welche zunächst den Zustand $\mathbf{z}_i$ und anschließend den Zustand $\mathbf{z}_j$ passiert haben, ist somit eine Teilmenge von $Paths(\mathbf{z}_i)$.

Da ein DTMC selbstverständlich die Markov-Eigenschaft der Gedächtnislosigkeit erfüllt (d.h., das zukünftige Verhalten hängt nur vom momentanen Zustand, nicht aber von den zuvor passierten Zuständen ab), lässt sich die Wahrscheinlichkeit $P(\mathbf{z}^\star, \omega)$ dafür, dass ein Zustand $\mathbf{z}^\star \in \mathbf{Z}$ über einen Pfad ω erreicht wird, als Produkt der durch die Zustandsübergangsmatrix $\mathbf{P}$ gegebenen Zustandsübergangswahrscheinlichkeiten $P_{\omega(i-1),\omega(i)}$ definieren. Hierbei stellt $\omega(i)$ den i-ten Zustand in der durch ω beschriebenen Folge dar und ist disjunkt zu $\omega(j)$, wobei $i \neq j$ gilt:

$$P(\mathbf{z}^\star, \omega) = \begin{cases} \prod_{i=1}^{\{j|\omega(j)=\mathbf{z}^\star\}} P_{\omega(i-1),\omega(i)} & \text{, falls } \exists j|\omega(j) = \mathbf{z}^\star \text{ und } j > 0 \\ 1 & \text{, falls } \omega(0) = \mathbf{z}^\star \\ 0 & \text{, sonst} \end{cases} \tag{3.1}$$

Die Wahrscheinlichkeit $\mathrm{Wsk}(\mathbf{z}^\star|\mathbf{z}^0)$, einen Zustand $\mathbf{z}^\star$ von einem Anfangszustand $\mathbf{z}^0$ aus zu erreichen, ergibt sich somit als Summe der Wahrscheinlichkeiten $P(\mathbf{z}^\star, \omega)$ aller von $\mathbf{z}^0$ nach $\mathbf{z}^\star$ führenden Pfade $Paths(\mathbf{z}^\star|\mathbf{z}^\mathbf{0})$:

$$\mathrm{Wsk}(\mathbf{z}^\star|\mathbf{z}^0) = \sum_{\omega \in Paths(\mathbf{z}^\star|\mathbf{z}^0)} P(\mathbf{z}^\star, \omega) \qquad \text{mit } \mathbf{z}^\star, \mathbf{z}^0 \in \mathbf{Z} \tag{3.2}$$

Analoges lässt sich formulieren, falls die Wahrscheinlichkeit gesucht ist, von einem Anfangszustand $\mathbf{z}^0$ eine aus mehreren (disjunkten) Elementen bestehende Zielmenge zu erreichen. Voraussetzung hierfür ist, dass die zugehörigen Pfade paarweise disjunkt sind, d.h. keines der Elemente der Zielmenge ist ein Element eines Pfades von der Anfangsmenge zu einem anderen Element der Zielmenge: $Paths(\mathbf{z}_{k1}|\mathbf{z}_0) \not\subset Paths(\mathbf{z}_{k2}|\mathbf{z}_0)\ \forall \mathbf{z}_{k1}, \mathbf{z}_{k2} \in$ der Zielmenge, $\mathbf{z}_{k1} \neq \mathbf{z}_{k2}$.

Ist die Wahrscheinlichkeit gesucht, eine Zielmenge von einer hierzu disjunkten Anfangsmenge aus zu erreichen, so lässt sich die Berechnung in drei Schritte unterteilen. Zunächst wird für jedes Element der Anfangsmenge die Wahrscheinlichkeit ermittelt, die Zielmenge von dort aus zu erreichen. Anschließend wird dieses Zwischenergebnis mit der Anfangswertwahrscheinlichkeit multipliziert. Unter der Anfangswertwahrscheinlichkeit versteht man hierbei die Wahrscheinlichkeit dafür, dass dieser Zustand als Anfangszustand ausgewählt wird. Schlussendlich erhält man die gesuchte Wahrscheinlichkeit durch Summation über die Teilwahrscheinlichkeiten.

Erweitert man die DTMC um eine Kosten- bzw. Ertragsstruktur $\mathbf{R} : \mathbf{Z} \times \mathbf{Z} \rightarrow \mathbb{R}_0^+$, so lässt sich eine Aussage über die mittleren Kosten eines Pfades treffen. Die Begriffe Kosten- und Erträge sind in diesem Zusammenhang als gleichwertig anzusehen. Wenn durch $R_{\omega(i-1),\omega(i)} \in \mathbf{R}$ die Kosten für einen Übergang vom (i-1)-ten zum i-ten Zustand eines Pfades ω gegeben sind, so sind – aufgrund der Markov-Eigenschaft „Gedächtnislosigkeit“ – die Kosten $R(\mathbf{z}^\star, \omega)$, um vom Anfangszustand $\mathbf{z}^0$ des Pfades ω zum Zustand $\mathbf{z}^\star$ zu gelangen, nach Gleichung 3.3 ermittelbar. Hierbei ist anzumerken, dass Kosten für das Erreichen eines (von $\mathbf{z}^0$) nicht-erreichbaren Zustandes als unendlich definiert sind.

$$R(\mathbf{z}^\star, \omega) = \begin{cases} \sum_{i=1}^{\{j|\omega(j)=\mathbf{z}^\star\}} R_{\omega(i-1),\omega(i)} & \text{, falls } P(\mathbf{z}^\star, \omega) > 0 \text{ und } \omega(0) \neq \mathbf{z}^\star \\ 0 & \text{, falls } \omega(0) = \mathbf{z}^\star \\ \infty & \text{, sonst} \end{cases} \tag{3.3}$$

Multipliziert man die Pfadkosten $R(\mathbf{z}^\star, \omega)$ (Gleichung 3.3) mit der Pfadwahrscheinlichkeit $P(\mathbf{z}^\star, \omega)$ (Gleichung 3.1) und summiert über alle vom (als gemeinsam angenommenen) Anfangszustand $\mathbf{z}^0$ zum Zustand $\mathbf{z}^\star$ verlaufenden Pfade $\Omega(\mathbf{z}_0^\star) := \mathit{Paths}(\mathbf{z}^\star|\mathbf{z}^0)$, so erhält man die mittleren Erreichungskosten

$$R(\Omega(\mathbf{z}_0^\star)) = \begin{cases} \sum_{\omega|\omega\in\Omega(\mathbf{z}_0^\star)} R(\mathbf{z}^\star, \omega) \cdot P(\mathbf{z}^\star, \omega) & \text{, falls } \sum_{\omega|\omega\in\Omega(\mathbf{z}_0^\star)} P(\mathbf{z}^\star, \omega) = 1 \\ \infty & \text{, sonst} \end{cases} \tag{3.4}$$

Entspricht die Kostenstruktur $\mathbf{R}$ eines DTMCs der auf $\{0, 1\}$ abgebildeten Zustandsübergangsfunktion $\mathbf{P}$ des DTMCs (1 = Übergang existiert, 0 = Übergang existiert nicht), so entsprechen die Kosten der Weglänge des Pfades. Bei Verwendung einer konstanten Zeitschrittweite Δt lässt sich hieraus direkt die zum Beschreiten dieses Pfades notwendige Zeit ermitteln (vgl. Abschnitt 3.1.2).

3.3.2 `Until`-Operator

Wie bereits ausgeführt, kann mittels des `Until`-Operators die Wahrscheinlichkeit dafür bestimmt werden, dass eine erste Bedingung mindestens so lange gilt, bis eine zweite Bedingung Gültigkeit erlangt hat, wobei die zweite Bedingung auch erfüllt werden muss. Man unterscheidet hierbei zwischen dem Time-Bounded `Until`-Operator, welcher die (durch Modellschritte repräsentierte, diskrete) Zeit, innerhalb der die zweite Bedingung erfüllt worden sein muss, beschränkt und dem stochastischen `Until`-Operator, der lediglich auf ein „irgendwann“ testet. Da die zugehörigen Berechnungsvorschriften nicht ganz trivial sind, aber ein Gefühl dafür vermitteln können, warum die PMC-Berechnungen teilweise durchaus länger dauern, sollen die Grundgedanken im Folgenden kurz angerissen werden.

Time-Bounded `Until`-Operator: $\mathtt{P} =? [\phi_1 \mathtt{U}^{\leq \mathtt{k}} \phi_2]$

Sei ϕ eine PCTL-Formel und $\vec{\Phi}_k,\ k \in \mathbb{N}_0$ ein Zustands-indizierter Spaltenvektor, der für jeden Anfangszustand $\mathbf{z}^0 \in \mathbf{Z}$ die, über alle von diesem Zustand abgehenden Pfade der

Länge k (entsprechend der Pfadwahrscheinlichkeiten) gewichtete, Wahrscheinlichkeit dafür angibt, dass ϕ nach k Schaltvorgängen des Automaten (mindestens einmal) erfüllt worden ist. Hierbei ist $\vec{\Phi}_0(\phi)(\mathbf{z}_i) = 1$, falls ϕ im Zustand $\mathbf{z}_i \in \mathbf{Z}$ zum Zeitpunkt 0 erfüllt ist und $\vec{\Phi}_0(\phi)(\mathbf{z}_i) = 0$, falls dies nicht der Fall ist.

Für die Anwendung dieses `Until`-Operators ist es erforderlich, zu bestimmen, ob

1. ϕ_1 von Anfang an erfüllt ist.
2. ϕ_2 irgendwann (jedoch innerhalb des betrachteten Zeitfensters) erfüllt sein wird und
3. ϕ_1 mindestens so lange erfüllt ist, bis ϕ_2 erfüllt ist.

Zur weiteren Untersuchung dieser Fragestellung wird die Zustandsmenge $\mathbf{Z}$ in drei disjunkte Teilmengen aufgeteilt, nämlich $\mathbf{Z}^{no}$, $\mathbf{Z}^{yes}$ und $\mathbf{Z}^{?}$:

$\mathbf{Z}^{yes}$ beinhaltet alle Zustände, für welche $\vec{\Phi}_0(\phi_2)(\mathbf{z}_i) = 1$ gilt und die somit $\mathtt{P} =?[\phi_1 \mathtt{U}^{\leq \mathtt{k}} \phi_2]$ trivialerweise erfüllen.

$\mathbf{Z}^{no}$ beinhaltet alle Zustände, die weder ϕ_1 noch ϕ_2 erfüllen, d.h. jene, für die gilt: $(\vec{\Phi}_0(\phi_1)(\mathbf{z}_i) = 0) \wedge (\vec{\Phi}_0(\phi_2)(\mathbf{z}_i) = 0)$.

$\mathbf{Z}^{?}$ beinhaltet alle übrigen Zustände, d.h. $\mathbf{Z}^{?} = \mathbf{Z} \setminus \{\mathbf{Z}^{yes} \cup \mathbf{Z}^{no}\}$.

Somit lässt sich die notwendige Berechnung auf die Rekursionsformel

$$\vec{\Phi}^{\star}_{k+1} = (\mathbf{P}^{\star}) \cdot \vec{\Phi}^{\star}_{k} \qquad \text{mit } \vec{\Phi}^{\star}_{0} = \vec{\Phi}_0,\ k \in \mathbb{N}^{+}_{0} \text{ und den } \mathbf{P}^{\star}\text{-Matrixelementen} \tag{3.5}$$

$$\mathbf{P}^{\star}(\mathbf{z}_j, \mathbf{z}_i) = \begin{cases} \mathbf{P}(\mathbf{z}_j, \mathbf{z}_i) & \text{, falls } \mathbf{z}_i \in \mathbf{Z}^{?} \text{ und } \mathbf{z}_i \neq \mathbf{z}_j \\ 1 & \text{, falls } \mathbf{z}_i \in \mathbf{Z}^{yes} \text{ und } \mathbf{z}_i = \mathbf{z}_j \\ 0 & \text{, sonst} \end{cases}$$

reduzieren, wobei $\mathbf{P}(\mathbf{z}_j, \mathbf{z}_i)$ die Übergangswahrscheinlichkeit vom Zustand $\mathbf{z}_i$ in den Zustand $\mathbf{z}_j$ darstellt.

Stochastischer `Until`-Operator: $\mathtt{P} =?\ [\phi_1 \mathtt{U} \phi_2]$

Für den stochastischen Until-Operator werden ebenfalls drei Teilmengen $\mathbf{Z}^{no}$, $\mathbf{Z}^{yes}$ und $\mathbf{Z}^{?}$ gebildet:

$\mathbf{Z}^{no}$ wird berechnet, indem zunächst die Menge all jener Zustände gebildet wird, von denen sich (mit einer Wahrscheinlichkeit größer 0) ein Zustand erreichen lässt, in dem ϕ_2 gilt, ohne einen Zustand zu passieren, in dem ϕ_1 nicht gilt. Anschließend werden $\mathbf{Z}^{no}$ all jene Elemente von $\mathbf{Z}$ zugeordnet, die sich nicht in dieser, die Erfüllung ermöglichenden Menge, befanden. $\mathbf{Z}^{no}$ enthält somit all jene Zustände, von denen aus es keine Möglichkeit gibt, die PCTL-Formel $\mathtt{P} =?[\phi_1 \mathtt{U} \phi_2]$ zu erfüllen.

$\mathbf{Z}^{yes}$ wird berechnet, indem zunächst die Menge all jener Zustände ermittelt wird, von denen es möglich ist, einen Zustand aus $\mathbf{Z}^{no}$ zu erreichen, ohne einen Zustand zu passieren, in dem ϕ_1 nicht, jedoch ϕ_2 gilt. Anschließend werden $\mathbf{Z}^{yes}$ all jene Elemente aus $\mathbf{Z}$ zugeordnet, die sich weder in dieser (gerade bestimmten Menge), noch in $\mathbf{Z}^{no}$ befinden. $\mathbf{Z}^{yes}$ enthält somit alle Zustände, welche die PCTL-Formel $\mathtt{P}=?[\phi_1 \mathtt{U} \phi_2]$ garantiert (d.h. mit einer Wahrscheinlichkeit von 100%) erfüllen.

$\mathbf{Z}^{?}$ beinhaltet schlussendlich alle übrigen Zustände, d.h. $\mathbf{Z}^{?} = \mathbf{Z} \setminus \{\mathbf{Z}^{yes} \cup \mathbf{Z}^{no}\}$.

Sowohl für die Berechnung von $\mathbf{Z}^{no}$, als auch $\mathbf{Z}^{yes}$ sind $|\mathbf{Z}|$ Iterationen des zugehörigen Algorithmus notwendig. Die gesuchten Wahrscheinlichkeiten ergeben sich durch Lösung des folgenden linearen Gleichungssystems, wobei $\vec{\Phi}_{\diamond}(\phi_1\ U\ \phi_2)(\mathbf{z}_i)$ die gesuchte Wahrscheinlichkeit $\mathtt{P}=?[\phi_1\ \mathtt{U}\ \phi_2]$ für den Zustand $\mathbf{z}_i$ darstellt:

$$\vec{\Phi}_{\diamond}(\phi_1\ U\ \phi_2)(\mathbf{z}_i) = \begin{cases} 0 & \text{, falls } \mathbf{z}_i \in \mathbf{Z}^{no} \\ 1 & \text{, falls } \mathbf{z}_i \in \mathbf{Z}^{yes} \\ \sum\limits_{\mathbf{z}_j \in \mathbf{Z}} \mathbf{P}(\mathbf{z}_i, \mathbf{z}_j) \cdot \vec{\Phi}_{\diamond}(\phi_1\ U\ \phi_2)(\mathbf{z}_j) & \text{, falls } \mathbf{z}_i \in \mathbf{Z}^{?} \end{cases} \tag{3.6}$$

Da die Wahrscheinlichkeiten für die Elemente der Mengen $\mathbf{Z}^{no}$ und $\mathbf{Z}^{yes}$ durch 0 bzw. 1 bereits vorab bekannt sind, lässt sich dieses Gleichungssystem von $|\mathbf{Z}|$ auf $|\mathbf{Z}^{?}|$ Unbekannte reduzieren:

$$\vec{\Phi}_{\diamond}(\phi_1\ U\ \phi_2)(\mathbf{z}_i) = \sum_{\mathbf{z}_j \in \mathbf{Z}^{?}} \mathbf{P}(\mathbf{z}_i, \mathbf{z}_j) \cdot \vec{\Phi}_{\diamond}(\phi_1\ U\ \phi_2)(\mathbf{z}_j) + \sum_{\mathbf{z}_j \in \mathbf{Z}^{yes}} \mathbf{P}(\mathbf{z}_i, \mathbf{z}_j) \tag{3.7}$$

4 Problem des Initialzustands

Voraussetzung für die Anwendung der PMC ist, die Initialzustandsmenge so gewählt zu haben, dass sämtliche Systemevolutionen der Länge eines Durchlaufs einen Startknoten in dieser Menge besitzen. Die korrekte, effiziente und vor allem vollständige Bestimmung der Anfangszustandsmenge ist jedoch keineswegs trivial und soll daher im Folgenden detailliert diskutiert werden.

Die zu betrachtenden Systeme bestehen aus mehreren komplexen Prozessen, die sowohl nebenläufig als auch in Abhängigkeit zueinander ausgeführt werden. Derart zusammengesetzte Systeme lassen sich am Besten mittels eines modularen Ansatzes modellieren, d.h. ein jeder Teilprozess wird zunächst eigenständig modelliert und im Zuge der Instanziierung mit den (minimal) notwendigen Kopplungen versehen. Bei der Ermittlung des Anfangszustands spielt eine spezielle Prozesseigenschaft eine besondere Rolle, nämlich die des im nächsten Abschnitt vorzustellenden „repetitiven“ Prozesses.

Im darauf folgenden Abschnitt 4.2 wird demonstriert, wie die Menge der Anfangszustände eines aus mehreren repetitiven Individualprozessen bestehenden Systems bestimmt wird. Abschnitt 4.3 stellt dann im Anschluss die Philosophie des Signal-Trackings vor. Signal-Tracking wird verwendet, um den Fortschritt eines Prozesses in einer abstrahierten Umgebung zu verfolgen. Hierzu wird in Abschnitt 4.4 diskutiert, welche Aspekte bei der Modellierung des Ereigniseintritts zu berücksichtigen sind. Den Abschluss dieses Kapitels (Abschnitt 4.6) bildet eine Vorstellung der Möglichkeiten, die diskutierten Anfangszustandsverteilungen in PRISM zu implementieren.

4.1 Repetitive Prozesse

Für die weitere Diskussion werden zunächst die Begriffe des repetitiven Automaten (und des hiermit modellierten repetitiven Prozesses), der Verweildauer und der Durchlaufzeit definiert.

Ein Automat wird genau dann als **repetitiv** (i.S.v. „sich wiederholend“) bezeichnet, wenn die folgenden Bedingungen erfüllt sind:

- Der Automat ist entweder autonom oder wird nur durch ein oder mehrere zufällige, gleichverteilte, unabhängige und externe (im Sinne von nicht-systeminherente) Signale beeinflusst.
- Jeder Zustand des Automaten hat mindestens einen Nachfolgezustand, welcher vom aktuellen Zustand aus in endlicher Zeit erreichbar ist.
- Für jeden Zustand gibt es zu jedem Zeitpunkt stets mindestens eine Möglichkeit, jeden anderen Zustand in endlicher Zeit zu erreichen. Das bedeutet, dass sich jeder Zustand in einer immer wieder erneut durchlaufenen Schleife befindet. Darüber hinaus muss es mindestens einen Zustand geben, der Teil aller im Automaten

existenten Schleifen ist. Die Zeit, welche notwendig ist, um von einem solch exponierten Zustand in denselben zurückzukommen, wird als **Durchlaufzeit** definiert. Existiert mehr als eine Schleife, ist die Durchlaufzeit des Automaten als die mit ihren Eintrittswahrscheinlichkeiten gewichtete Summe der Durchlaufzeiten aller Schleifen definiert.

– Zu jedem zufällig gewählten Zeitpunkt t ist die Wahrscheinlichkeit dafür, dass sich der Automat in einem bestimmten (aber beliebigen) Zustand $\mathtt{z_k}$ befindet, größer Null: $p(\mathtt{z_k}, t) > 0$.

Betrachtet man einen repetitiven Prozess zu einem zufälligen Zeitpunkt, so entspricht folglich die Wahrscheinlichkeit $p(\mathtt{z_k}, t)$ dafür, dass sich der Automat in einem bestimmten Zustand $\mathtt{z_k}$ befindet, dem Quotienten aus der mittleren Verweilzeit dieses Zustands und der Durchlaufzeit. Diese Folgerung begründet sich auf den folgenden beiden Annahmen:

1. Annahme: Die Systemzeit t schreitet in immer gleicher Weise voran. Die Ableitung lokaler Uhren nach t ist Eins für aktivierte, und Null für nicht aktivierte Uhren (vgl. Kapitel 5.3).
   Eine hinreichende Bedingung zur Erfüllung dieser Annahme umfasst ein unterbrechungsfreies System und eine feste Zeitbasis für t.
2. Annahme: Für jeden Zustand ist die Verteilung der Zustandsübergangsauslösezeitpunkte bekannt. Hieraus lässt sich die, als Erwartungswert dieser Verteilung definierte, mittlere **Verweilzeit** ebenso ermitteln, wie die Durchlaufzeit.

Da in der Praxis die Schaltzeitpunkte vieler repetitiver Prozesse von ihren Nominalwerten abweichen oder um dieselben schwanken, muss entweder garantiert werden, dass die Abweichung vom Nominalwert deutlich unterhalb der (zeitlichen) Abbildungsgenauigkeit ist – in diesem Fall kann mit einem konstanten Schaltzeitpunkt gearbeitet werden – oder die auftretenden Abweichungen müssen mittels stochastischer Zeitverteilungen beschrieben werden. Automatenzustände, deren abgehende Übergänge eine stochastisch beschriebene Verweilzeit aufweisen, werden entsprechend ihrer mittleren Verweilzeit berücksichtigt. Dies soll anhand der folgenden beiden Beispiele verdeutlicht werden:

A. Gegeben sei ein aus den drei Zuständen `init`(ialisieren), `bearbeiten` und `ausgeben` bestehender repetitiver Prozess. Während sich dieser Prozess in den Zuständen `init` und `ausgeben` jeweils genau 3 Zeiteinheiten aufhält, beträgt die Verweilzeit im Zustand `bearbeiten` entweder 3 oder 5 Zeiteinheiten. Die Zeitspannen 3 und 5 seien gleich wahrscheinlich, d.h., die Wahrscheinlichkeit, dass die Verweilzeit im Zustand `bearbeiten` 3 Zeiteinheiten beträgt entspricht derjenigen, dass sie 5 Zeiteinheiten beträgt – und ist somit jeweils genau 0,5. Gewichtet man die beiden Verweilzeiten mit ihren Eintrittswahrscheinlichkeiten, ergibt sich eine mittlere Verweilzeit im Zustand `bearbeiten` von 4 Zeiteinheiten. Somit lässt sich die Durchlaufzeit zu 10 (Zeiteinheiten) bestimmen. Die Division der mittleren Verweilzeiten mit der Durchlaufzeit ergibt die Aufenthaltswahrscheinlichkeiten in den drei Zuständen: 30%, für die Zustände `init` und `ausgeben`, sowie 40%, für den Zustand `bearbeiten`.

B. Als deutlich komplexeres Beispiel sei ein aus den beiden Zuständen `untätig` und `arbeitend` bestehender zeitdiskreter Automat gegeben. Befindet sich dieser Automat im Zustand `untätig`, so kann er Arbeitsaufgaben annehmen und in den Zustand `arbeitend` wechseln. Die, auf die Länge der diskreten Zeitschrittweite bezogene Wahrscheinlichkeit für einen solchen Wechsel, betrage p_A. Nach der T_A Zeitschritte erfordernden Fertigstellung des Auftrages kann entweder sofort ein neuer Auftrag angenommen werden (die Wahrscheinlichkeit hierfür betrage ebenfalls p_A) oder in den Zustand `untätig` gewechselt werden. Dies ist in Abbildung 12 schematisch dargestellt. Mit diesen Angaben lässt sich die Wahrscheinlichkeit für einen Aufenthalt im Zustand `untätig` analytisch zu

$$p_u = \frac{1 - p_A}{1 + (T_A - 1) \cdot p_A} \text{ , mit } T_A \in \mathbb{N} \tag{4.1}$$

bestimmen. Die Wahrscheinlichkeit für einen Aufenthalt im Zustand `arbeitend` ergibt sich zu $p_w = 1 - p_u$. Schlussendlich lässt sich die Wahrscheinlichkeit dafür, dass sich der Automat zu einem beliebig gewählten Zeitpunkt gerade im Zustand `arbeitend` befindet und noch für $\tilde{x}$ Zeitschritte in diesem Zustand sein wird, bestimmen:

$$\frac{\tilde{x} \cdot p_w}{T_A} \text{ , mit } T_A, \tilde{x} \in \mathbb{N} \text{ und } \tilde{x} < T_A \tag{4.2}$$

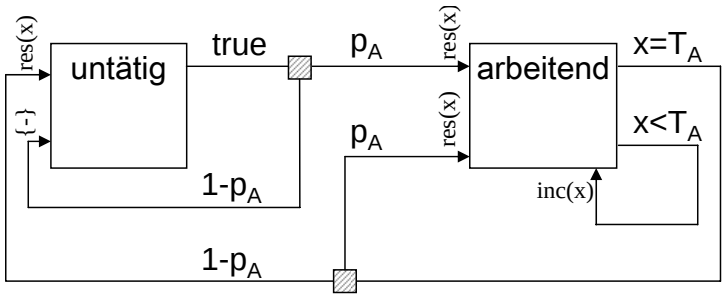

Abbildung 12: Diskreter Automat eines einfachen Maschinenprozesses. Die hierbei verwendete Automatennotation wird in Kapitel 5.4 erläutert.

Hinweis: Soweit nicht anders vermerkt, wird der Terminus „zufällig“ im Rahmen dieses Buches mit Weißem Rauschen assoziiert. Weißes Rauschen lässt sich für einen ausreichend großen Stichprobenumfang auf eine Gleichverteilung zurückführen. Die Herleitung anderer Verteilungen ist zwar verhältnismäßig einfach, allerdings ist die Implementierung derselben deutlich aufwändiger, als diejenige des weißen Rauschens.

Im Gegensatz zu den repetitiven Prozessen, bei denen es möglich ist, den Prozess so zu transformieren, dass er durch ein Spektrum wahrscheinlichkeitsgewichteter Automaten (bzw. -pfade) dargestellt werden kann, ist dies für die meisten der nicht-repetitiven Prozesse nicht möglich. Da die Mehrzahl von ihnen das für die Klassifikation als repetitiver Prozess erforderliche Kriterium der Autonomie verletzt, müssen diese Prozesse von dem durch andere Prozesse erzeugten Startzeitpunkt an berücksichtigt werden. Das bedeutet,

dass der Startzeitpunkt i.d.R. nicht als Anfangsverteilung angebbar ist, sondern aufgrund der Evolution des Gesamtsystems bestimmt wird.

Anmerkung: Selbstverständlich lässt sich die mittlere Verweilzeit auch für einzelne Zustände nicht-repetitiver Automaten ermitteln, falls deren abgehende Übergänge lediglich rein zeitbasierte Bedingungen aufweisen.

4.2 Verkopplung mehrerer repetitiver Prozesse

Nimmt man für jeden Einzelprozess die Annahmen 1 und 2 als gegeben an, stellt sich die Frage, wie die Anfangszustände mehrerer repetitiver Prozesse in Relation zueinander stehen.

Unter der Voraussetzung, dass die Einzelprozesse vollständig unabhängig voneinander sind, lässt sich der Anfangszustandsvektor aus der Menge aller möglichen Kombinationen bilden, die sich aus den Anfangszustandsmengen der Einzelprozesse ergeben. Hieraus folgt, dass derartige Einzelprozesse mittels unabhängiger Module modelliert werden können, da PRISM in diesem Fall alle möglichen Kombinationen der Anfangszustandsmengen automatisch erzeugt. Beispiel: Besitzt ein Prozess A fünf, ein Prozess B vier und ein Prozess C neun Zustände, so umfasst die Menge möglicher Kombinationen 180 Elemente. Die zugehörigen Wahrscheinlichkeiten ergeben sich durch Multiplikation der Anfangszustandswahrscheinlichkeiten der Einzelprozesse: Seien A= $\{a_1, \ldots, a_5\}$, B= $\{b_1, \ldots, b_4\}$ und C= $\{c_1, \ldots, c_9\}$ sowie $p_A^0(a_1) = 36\%,\ p_A^0(a_i,\ i > 1) = 16\%,\ p_B^0(b_j) = \frac{1}{4}$ und $p_C^0(c_k) = \frac{1}{9}$ so ergibt sich die Wahrscheinlichkeit dafür, dass der Prozess mit dem Tupel $\{a_i, b_j, c_k\}$ startet zu 1%, falls $i = 1$ und zu 0,44%, falls $i > 1$.

Neben diesem Spezialfall, in dem keinerlei Kopplung zwischen den Einzelprozessen auftritt, gibt es eine ganze Reihe mehr oder weniger gekoppelter Prozesse. Solch eine Kopplung wird nicht nur durch eine direkte Abhängigkeit induziert, sondern tritt auch im Falle linear abhängiger Durchlaufzeiten oder dem gleichzeitigen Start von mehreren Teilprozessen auf. Hieraus ergeben sich fünf zu berücksichtigende Phänomene indirekter Kopplung.

Hinweis: Die folgende Analyse der Durchlaufzeiten beruht auf der Verwendung von diskreten Zeitschritten ($\in \mathbb{N}$). Zwei Durchlaufzeiten werden im Rahmen dieser Arbeit genau dann als linear unabhängig angesehen, wenn ihre, auf volle Zeitschritte aufgerundeten Durchlaufzeiten teilerfremd sind. Darüber hinaus muss beachtet werden, dass zwei Prozesse, die eine identische Durchlaufzeit aufweisen, nicht identisch sein müssen. Unterschiede können sich z.B. in Bezug auf die Anzahl der Zustände, die Anfangszustandsverteilung oder die Zustandsübergangsmatrizen finden.

1. Haben die unabhängigen Prozesse identische (und synchronisierte) Durchlaufzeiten und wurden gleichzeitig gestartet, so sind diese de facto gekoppelt und müssen auch so implementiert werden. Haben alle Prozesse die gleiche Startzeitstruktur, bedeutet dies, dass die Abstraktion der Einzelprozesse jeweils auf den gleichen abstrakten Automaten führt. Der aus der Überlagerung entstehende Gesamtautomat besitzt

im allgemeinen Fall mindestens einen gemeinsamen Zustand aller Prozesse. Hierbei handelt es sich um jenen Zustand, von dem aus die Prozesse gestartet wurden und der jeweils nach einem Durchlauf wieder erreicht wird. Hiervon abgesehen spiegelt der Gesamtautomat sowohl das Verhalten des Einzelnen als auch des Gesamten wider.

2. Die unabhängigen Prozesse haben identische (und synchronisierte) Durchlaufzeiten, aber unterschiedliche Startzeitpunkte: Wurden die einzelnen Prozesse mit einer festen Zeitdifferenz gestartet, lässt sich dieser Fall auf Punkt 1 zurückführen. Sind die Startzeitpunkte hingegen zufällig über eine Zeitspanne verteilt, die (deutlich) größer ist als die Durchlaufzeit, kann das System als zufällig bezeichnet werden. Hieraus folgt, dass der Anfangszustandsvektor aus der Menge aller möglichen Anfangszustände der Einzelprozesse zusammengesetzt werden kann, d.h., dass die zugehörigen Anfangszustandswahrscheinlichkeiten einer Gleichverteilung unterliegen.

   Leider ist der Fall, bei dem die zufälligen Zeitdifferenzen kleiner sind als die Durchlaufzeit, deutlich schwieriger. Tritt dieser ein, müssen all jene Kombinationen, welche sich innerhalb der Überlappungszone befinden, berücksichtigt werden, während jene, die sich außerhalb befinden, nicht berücksichtigt werden dürfen. Beispielhaft seien zwei Prozesse mit einer Durchlaufzeit von jeweils 5 Zeitschritten betrachtet, wobei zur Vereinfachung angenommen wird, dass beide Prozesse pro Zeitschritt genau einen Zustandswechsel ausführen. Wird der zweite Prozess entweder einen oder zwei Zeitschritte nach dem ersten Prozess gestartet, ergeben sich (aus der Sicht eines zu einem zufälligen Zeitpunkt in Erscheinung tretenden Beobachters) die folgenden 10 möglichen Zustandskombinationen: (2,1), (3,2), (4,3), (5,4) und (1,5) sowie (3,1), (4,2), (5,3), (1,4) und (2,5). Treten die beiden Verschiebungen jeweils mit einer Wahrscheinlichkeit von 50% auf (d.h. der Eintritt beider Verschiebungen ist gleichwahrscheinlich), weisen alle zehn möglichen Anfangskombinationen eine Eintrittswahrscheinlichkeit von 10% auf. Wenn jedoch eine Zeitverschiebung um 2 Zeitschritte dreimal so wahrscheinlich ist, wie diejenige um 3 Zeitschritte, weisen die ersten fünf der oben genannten zehn Kombinationen eine Eintrittswahrscheinlichkeit von 5% und die verbleibenden von 15% auf.

3. Bei gleichzeitigem Start mehrerer Prozesse mit linear abhängigen Durchlaufzeiten treten nicht zwingend alle theoretisch möglichen Kombinationen der einzelnen Anfangszustandsmengen auf. Beispielhaft seien Kombinationen der Durchlaufzeiten 5 und 10 oder 6 und 8 genannt: Im ersten Fall (5 und 10) treten lediglich die Kombinationen (5,5) sowie (5,10) auf, während im zweiten Fall (6 und 8) alle Kombinationen der Vielfachen von zwei auftreten. Um dies zu modellieren, ist ein allgemeiner Gesamtautomat, wie er bereits für den 1. Fall erläutert wurde, für jede der möglichen Anfangszustandskombinationen notwendig.

4. Linear abhängige Durchlaufzeiten mit unterschiedlichen Startzeitpunkten: Für diesen Fall gelten dieselben Überlegungen wie für die vorherigen beiden. Er wird daher nicht weiter aufgeschlüsselt.

5. Der Fall linear unabhängiger Durchlaufzeiten mit gleichzeitigem Start mehrerer Prozesse ist der wohl ungünstigste, da analytisch geprüft werden muss, ob wirklich alle (oder ggf. welche) der theoretisch möglichen Kombinationen tatsächlich auftreten. Im günstigsten Fall lässt sich dies auf den idealen Fall (keinerlei Kopplungen) zurückführen. Andernfalls muss die Gesamtheit aller auftretenden Kombinationen des allgemeinen Gesamtautomaten entsprechend berücksichtigt werden.

Für die weitere Diskussion wird der bereits angesprochene Idealfall verwendet, bei dem keinerlei Abhängigkeiten der Prozesse untereinander existieren (linear unabhängige Durchlaufzeiten mit zufälligen Startzeitpunkten).

Anmerkung: Vergrößert man jede einzelne Verweilzeit um eine zufällige, reellwertige Zeitdauer ($\tau \ll$ Diskretisierungsschrittweite) kann – nach einer ausreichend langen Einschwingzeit – angenommen werden, dass die Anfangszustände zufällig verteilt sind. Dies wird jedoch in der weiteren Diskussion nicht berücksichtigt.

4.3 Signal-Tracking (Signalbeobachtung)

Zur Bestimmung von Antwortzeiten ist es erforderlich den Startzeitpunkt ebenso zu erfassen, wie das sich aus dem Prozessablauf ergebende Beobachtungsende. Im Unterschied zu einer Simulation, bei der man alle im System ablaufenden Prozesse und Signale mit eindeutigen Teilinformationen versehen kann und somit lediglich abwarten muss, bis ein entsprechend gekennzeichnetes Signal das gewünschte Ausgangsverhalten auslöst, ist dies bei PMC – aus Gründen des beschränkten Zustandsraumes – nur in sehr begrenztem Umfang möglich. In vielen Fällen ist es jedoch gar nicht erforderlich, alle Funktionalitäten einer Komponente explizit zu modellieren. Dies ist z.B. der Fall, wenn ihr Ergebnis nur dann von Interesse ist, wenn es sich im Fortfluss des einen betrachteten Eingangssignalwechsels befindet und ansonsten vernachlässigt werden kann. Aus diesem Grunde beschränkt man sich auf die Modellierung der tatsächlich wichtigen Signalverläufe in expliziter Form. Signale, welche das Systemverhalten nicht verändern, werden hingegen weggelassen oder – so sie die gesuchten Prozesse beschreiben – mittels Signalbeobachtungsmodulen quasi als Überprozess auf der ansonsten autonomen Systemstruktur modelliert.

Signal-Tracking ist dabei die Kunst, die fallspezifischen Informationen (Attribute) aus den Einzelprozessen auszulagern und als getrennte Module zu implementieren. Dies eröffnet die Möglichkeit, nur jene fallspezifischen Informationen zu implementieren, die für die gerade zu prüfende Eigenschaft notwendig sind. Signalbeobachtungsmodule ergeben sich also aus den zu observierenden Eigenschaften und tragen zur „Protokollierung“ des zur Erfüllung derselben führenden Prozesses bei. Mittels Signal-Tracking wird somit auch die in Kapitel 3 diskutierte Terminierung der Systemevolution sichergestellt. Ihre besondere Erfordernis resultiert schlussendlich jedoch auch daraus, dass PCTL (in der von PRISM verwendeten Syntax) die Beobachtung einer speziellen Zustandsabfolge nicht unterstützt.

Zur Verdeutlichung sei das in Abbildung 13 gezeigte Beispiel angeführt. Dieses bestehe eingangsseitig aus einem externen zeitlich gleichverteilten Ereignis, welches einem ersten

Prozess (#1) als Eingangsgröße zugeführt sei. Nach Verarbeitung durch diesen ersten nicht-repetitiven Prozess wird das hierbei erzeugte Ausgangssignal am Eingang eines zweiten Prozesses (#2) aktiv. Aufgrund der repetitiven Grundstruktur des Prozesses #2 muss dieses Signal warten, bis der Automat zur Weitergabe der Information bereit ist. Die Bereitschaft zur Weitergabe eines Signals ist hierbei als Aufenthalt in einem entsprechend gekennzeichneten Zustand zu verstehen. Der Prozess #3 ist wiederum nicht-repetitiv, d.h., sobald Prozess #2 die Information weitergegeben hat, was zyklisch geschieht, wird diese verarbeitet und die entsprechende Prozessausgabe erzeugt. Da Prozess #2 über der Zeit gesehen den entsprechenden Eingang aufgrund seines repetitiven Charakters immer wieder, jedoch i.d.R. mit unterschiedlichen Informationen ausgestattet, aktiviert, ist primär nur jene Ausgabe von Interesse, die als Folge des Ereigniseintritts an Prozess #1 angesehen werden kann.

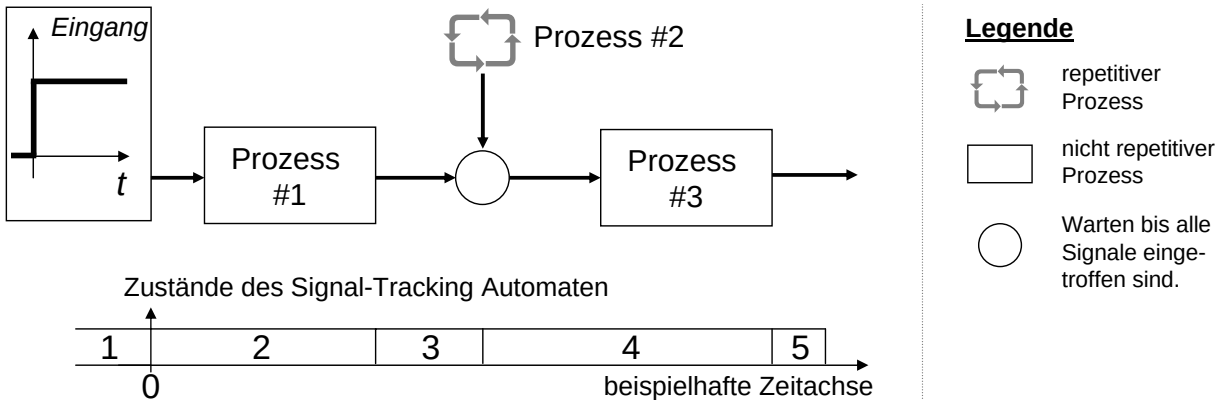

Abbildung 13: Einführendes Beispiel zum Übergang vom Prozessablauf auf das Signal-Tracking

Um dies zu modellieren sind vier Module erforderlich: Das erste und dritte beschreiben die nicht-repetitiven Prozesse #1 und #3, welche nach Eintreffen eines Eingangssignals ihren Ablauf beginnen. Das zweite Modul beschreibt den repetitiven Prozess #2. Als viertes Modul findet sich das Signal-Tracking, welches das folgende Verhalten aufweist:

1. Starte, sobald das externe Ereignis eintritt (Zeit 0)
2. Warte, bis der erste Prozess fertig ist
3. Warte, bis der zweite Prozess den Zustand „Information weitergeben“ verlässt
4. Warte, bis der dritte Prozess fertig ist
5. Beende die Berechnung

Für die Implementierung bedeutet dies, dass Prozess #3 immer dann startet, wenn Prozess #2 den Zustand „Information weitergeben“ verlässt, unabhängig davon, ob die vom Prozess #2 übermittelte Information für das zu beobachtende Verhalten wichtig ist. Diese Vorgehensweise vereinfacht nicht nur die Modellierung, sondern reduziert auch den Speicher- und Rechenzeitbedarf. Darüber hinaus ermöglicht es dem Benutzer, das vom Prozess #3 somit zyklisch generierte Ausgangssignal anderweitig zu verwenden, wie z.B. zur Ansteuerung eines, vom Signal-Tracking erst im Resultat erfassten, Querprozesses. Diese Überlegungen gelten hierbei nur für den Prozess #3, nicht jedoch für den Prozess #1, da letzterer nur zum Zeitpunkt 0 und nicht in zyklischen Abständen angesteuert wird. Insofern wäre es möglich,

diesen ersten Prozess nur indirekt, d.h. als Teil des Signal-Trackings, zu implementieren, was zu einer geringen Reduktion des Speicher- und Rechenzeitbedarfs, aber leider auch zu Einbußen bei der Übersichtlichkeit, führen würde.

Ganz im Gegensatz hierzu macht es keinen Sinn, Prozess #2 erst zu starten, wenn Prozess #1 die Bearbeitung abgeschlossen hat, da die, in Bezug auf die Anzahl möglicher von der Anfangsmenge zur Ergebnismenge führenden Pfade, erwartbare Verringerung des Resourcenverbrauchs komplett durch die erhöhte Komplexität des Systemmodells aufgehoben wird (Prozess #2 wäre dann nicht mehr als unabhängiges Modul anlegbar). Darüber hinaus würde sich der für die Zustandsinformationen erforderliche Speicherplatz nicht verändern, da die Variablen des Prozesses #2 im Modell angelegt werden müssen, unabhängig davon, wann sie benötigt werden. Aus diesem Grunde lässt man repetitive Prozesse wie #2, von Beginn an und – von ganz wenigen Sonderfällen abgesehen – bis zum Ende der Berechnungen mitlaufen. Dies ermöglicht es außerdem, die von Prozess #2 erzeugten Informationen zu jeder Zeit durch andere – möglicherweise auch erst später hinzugefügte – Prozesse zu nutzen, ohne hierfür die Struktur verändern zu müssen.

4.4 Modellierung des Ereigniseintritts

Wie in Abschnitt 3.2.3 erläutert, ist es notwendig, eine Anfangszustandstransformation durchzuführen. Während sich repetitive Prozesse auf eine Multimenge von mit Aufenthaltswahrscheinlichkeiten gewichteten repetitiven Prozessen transformieren lassen, ist dies für nicht-repetitive Prozesse nicht möglich. Dies liegt meist an der Abhängigkeit von anderen Signalen. Da die hierfür erforderliche Information nicht im Voraus bereitgestellt werden kann, ist es notwendig, die Zeitachse der Berechnung so zu verschieben, dass der Ereigniseintritt erst nach dem Ablauf einer Einschwingzeit auftritt. Die eigentliche Herausforderung ist es dabei, die notwendige Einschwingzeit möglichst genau zu bestimmen, da aufgrund unterschiedlicher Einflussfaktoren, wie z.B. Speichergröße, Prozessortakt, Beschränkungen durch die PMC-Software oder der für die Berechnung erforderliche Zeitbedarf, diese Größe nicht beliebig groß gewählt werden darf. Hieraus folgt, dass der Ereigniseintritt zum frühest möglichen Zeitpunkt im Ablauf vorgesehen werden sollte. In diesem Unterkapitel wird daher erläutert werden, wie diese Grenze ausgelotet werden kann. Hierzu wird die Position des in Abbildung 13 eingeführten Prozesses #1, wie in Abbildung 14 gezeigt, verändert. Durch diese Änderung weist das System nun einen Haupt- und einen Nebenarm auf, was bedeutet, dass das von Prozess #2 erzeugte Signal zunächst den Prozess #1 passieren muss, bevor eine Synchronisation mit dem vom externen Ereignis erzeugten Signal vorgenommen werden kann.

Das zugehörige Signal-Tracking lässt sich zwar aus den folgenden vier Schritten zusammensetzen,

1. Starte, sobald das externe Ereignis eintritt (Zeit 0)
2. Warte, bis das vom Prozess #1 erzeugte Signal eingetroffen ist
3. Warte, bis der Prozess #3 fertig ist
4. Beende die Berechnung

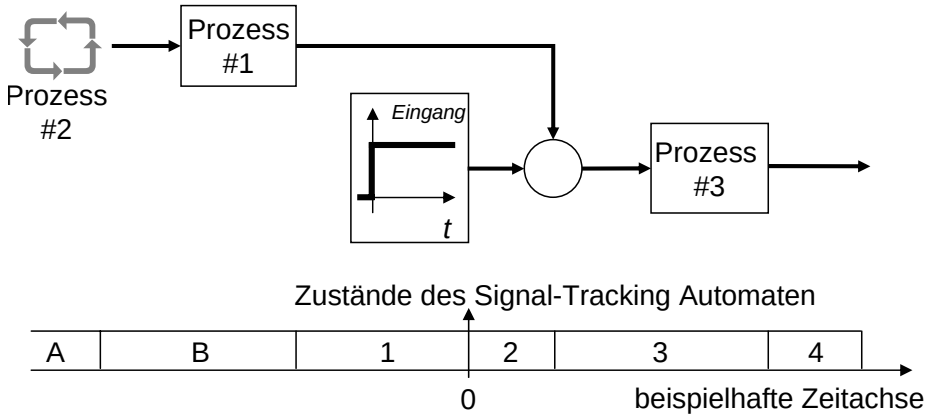

Abbildung 14: Prozessablauf eines Systems mit kaskadiertem Quersignal

nicht geklärt ist damit jedoch, wie mit dem Nebenarm verfahren werden soll. Muss das Signal-Tracking um die Zustände A und B (vgl. Abb. 14) erweitert werden? Wenn Prozess #1 eine rein deterministische (also feste) Durchlaufzeit besitzt, es sich also um eine bekannte Verzögerung handelt, deren einzige Folge es ist, die Zeitachse des Prozesses #2 entsprechend zu verschieben, dann ist dies nicht erforderlich. Diese konstante Verschiebung wird nämlich durch den Transformationsprozess (der Zeitachse auf den Ereigniseintritt) eliminiert, was bedeutet, dass Prozess #1 in den Betrachtungen vollständig vernachlässigt werden kann. Allerdings gilt dies nur, solange es sich bei der hervorgerufenen Verzögerung um eine bekannte und rein deterministische Größe handelt. Ist dies nicht der Fall oder kann die Information während der Bearbeitung durch Prozess #1 verloren gehen, ist eine detailliertere Analyse notwendig.

4.4.1 Nicht-repetitive Prozesse mit stochastischer Durchlaufzeit

Aus Sicht des Signal-Trackings ist es im Falle des in Abbildung 13 gezeigten Aufbaus unbedeutend, ob Prozess #1 das Signal verzögert, da das Signal-Tracking nur warten muss, bis dieser Prozess abgeschlossen ist. Anders stellt sich die Sachlage jedoch für den in Abbildung 14 gezeigten Aufbau dar. Zur Erläuterung sei zunächst angenommen, dass die Durchlaufzeit des Prozesses #2 derart kurz ist, dass Prozess #1 in jedem diskreten Zeitintervall eine Triggerung bekommt, wobei es (für dieses Beispiel) möglich sein soll, dass mehrere Prozess #1 Instanzen gleichzeitig aktiv sind, ohne dass sie sich gegenseitig beeinflussen. Da Prozess #1 darüber hinaus keine weiteren Eingangssignale verarbeitet, lässt sich (die Kontinuität der Prozesse vorausgesetzt) die Zustandsverteilung analytisch berechnen. Hieraus folgt, dass sich das Verhalten von Prozess #1 in diesem Fall ähnlich einem repetitiven Prozess auf die Ereigniseintrittszeitachse transformieren lässt. Für jedes in den Prozess #1 eingeschleuste Signal ist es daher möglich, zu bestimmen, mit welcher Wahrscheinlichkeit die zu diesem Eingangssignal gehörende Ausgabe ein, zwei, drei ... Zeitschritte nach der Aktivierung des Eingangs erfolgt. Für das folgende Gedankenexperiment sei angenommen, dass es sich hierbei nicht um Wahrscheinlichkeiten, sondern um Häufigkeiten handelt, d.h. von einer gewissen Anzahl (z.B. 1.000.000) an Eingangssignalen würde ein,

dieser Häufigkeit entsprechender Anteil um ein, zwei, drei ... Zeitschritte verzögert. Liegen nun in jedem Zeitschritt 1.000.000 Eingangssignale an, die jeweils eine um ein, zwei, drei ... Zeitschritte verzögerte Ausgabe zur Folge haben, so lässt sich feststellen, dass in jedem Zeitschritt auch 1.000.000 Ausgangssignale erzeugt werden und zwar unabhängig von der gewählten Häufigkeitsverteilung: Für jedes Signal (hier als abzählbares Ereignis verwendet), das z.B. um zwei Zeitschritte verzögert wurde und somit im – einer Verzögerung von einem Zeitschritt entsprechenden – Ausgangszeitschlitz fehlt, existiert genau ein Signal, das einen Zeitschritt früher in das System eingetreten ist, ebenfalls um zwei Zeitschritte verzögert wurde und folglich nun die durch die vermeintliche Verzögerung erzeugte Lücke wieder schließt. Solange also nur wichtig ist, dass ein Ausgangssignal erzeugt wird und nicht wann es in das System eingetreten ist, spielt bei gleichbleibender Eingangssituation das Verzögerungsverhalten dieses Prozesses für seinen Ausgang keine Rolle.

Tritt jedoch das Eingangssignal nicht mehr in jedem Zeitschritt auf, sondern z.B. nur alle 10 Takte, ist eine genauere Betrachtung erforderlich. Zunächst sei davon ausgegangen, dass das vom Prozess #1 erzeugte Signal nur eine Einkopplung in das Gesamtsystem habe, d.h. es keine Rolle für das System spielt, wann sich der Prozess, nachdem das betreffende Signal erzeugt worden ist, in welchem Zustand befindet. Geht man darüber hinaus davon aus, dass eine Aktivierung des Eingangssignals über der transformierten (kontinuierlichen) Zeitachse als Gleichverteilung auftritt, lässt sich obiges Gedankenexperiment erweitern: Nimmt man an, es lägen alle 10 Takte einmalig 10.000.000 Eingangssignale an, so ist die am Ausgang pro Takt zu erwartende Anzahl an Signalantworten weiterhin mit 1.000.000 anzugeben; und zwar wiederum unabhängig von der die Verzögerung bestimmenden Häufigkeitsverteilung. Dies liegt daran, dass aus Sicht des Ereigniseintrittspunktes die Wahrscheinlichkeit dafür, dass der Sendetakt in den betrachteten Zeitschlitz fällt, 10% beträgt. Allgemein lässt sich also festhalten, dass das transformierte Erscheinungsbild eines repetitiven Prozesses durch einen (nur von diesem repetitiven Prozess abhängigen) nicht-repetitiven Prozess mit stochastischer Durchlaufzeit nicht verändert wird, falls obige Annahmen erfüllt sind. Für die Bestimmung der Anfangszustandsverteilung ist ein solcher zwischengeschalteter Prozess dementsprechend ohne Einfluss. Hierdurch erklärt sich auch, warum es für die Frage des Initialzustands keine Rolle spielt, ob die Durchlaufzeit eines repetetiven Prozesses eine aus stochastischen Anteilen zusammengesetzte Kenngröße oder eine Konstante ist.

Dies gilt jedoch nicht mehr, sobald ein zukünftiger Zustand des Prozesses #2 für den weiteren Systemablauf entscheidend ist, wie beispielhaft in Abbildung 15 gezeigt. Dabei spielt es zunächst keine Rolle, ob die vom Prozess #2 kommende Information in beiden Fällen den Prozess #1 passieren muss, oder ob für die zweite Passage ein anderer Prozess anstelle von Prozess #1 zum Tragen kommt (z.B. #4).

Das Problem hierbei ist, dass die zweite Einkopplung nicht mehr die Gleichverteilung des Ursprungsprozesses aufweist, was bedeutet hätte, dass die entstehende Verzögerung eine stochastische (hier gleichverteilte) Funktion darstellen würde. Stattdessen ergibt sich die am zweiten Einkopplungspunkt entstehende Verzögerung als Funktion der Durchlaufzeit des repetitiven Prozesses #2 sowie der durch den Prozess #3 ins System gebrachten

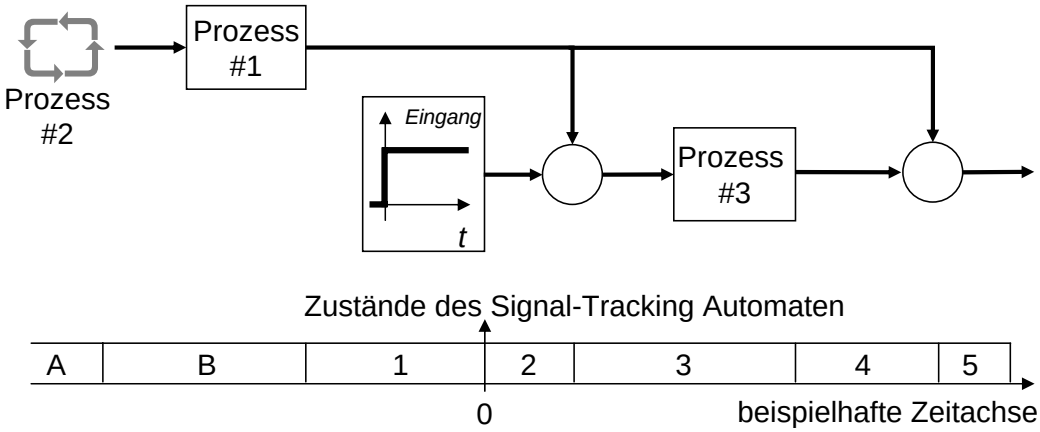

Abbildung 15: Prozessablauf eines Systems mit zweifach einkoppelndem Quersignal

Verzögerungen. Im Falle eines 4. Prozesses (#4) müsste darüber hinaus auch noch Prozess #1 und #4 berücksichtigt werden. Das Auftreten einer solchen zweiten Einkopplung erzwingt somit die Berücksichtigung der Zustände A und B im Rahmen des Signal-Trackings.

Um die beiden Zustände A und B zu implementieren, gibt es zwei Herangehensweisen: Die erste schätzt die realistisch betrachtet maximal notwendige Zeit, um diese beiden Phasen abzuarbeiten, und überlässt es dem System, sich innerhalb dieser Zeit entsprechend einzuschwingen. Im vorliegenden Fall würde man hierfür die maximale Durchlaufzeit des repetitiven Prozesses #2 nehmen und zu der maximalen Verzögerungsdauer des Prozesses #1 addieren. Existiert für die Verzögerungsdauer des Prozesses #1 keine obere Schranke (dies ist z.B. bei Exponentialfunktionen der Fall), verwendet man stattdessen einen Wert, der nur in z.B. 0,00001% der Fälle überschritten wird. Die zweite Herangehensweise invertiert die beteiligten Prozesse (hier: Prozess #1) und integriert ihr Verhalten explizit im Signal-Tracking. In diesem Fall ist es dann auch möglich, die Anfangswertbestimmung des Prozesses #2 mit dem Signal-Tracking zu koppeln. Kennt man im Beispiel der Abbildung 15 sowohl die Inverse des Prozesses #1 als auch die Anfangsverteilung des Prozesses #2, dann ist es möglich, Prozess #2 nicht nur in Abhängigkeit des Ereigniseintrittzeitpunktes zu initialisieren, sondern zusätzlich auf Basis des rückgerechneten Verhaltens von Prozess #1. In anderen Worten bedeutet dies, dass hiermit die beteiligten Prozesse nicht mehr getrennt betrachtet werden könnten, sondern gemeinsam behandelt werden müssten. Obwohl dies für ein einfaches System wie das gezeigte durchaus machbar ist, erschwert sich hierdurch nicht nur der Entwurf, sondern es eröffnet sich auch eine zusätzliche Fehlerquelle.

Aus diesem Grunde wird man im Normalfall, insbesondere wenn man es mit komplexen Systemen zu tun hat, die erste Herangehensweise verwenden. Hierbei werden die Anfangswerte aller repetitiven Prozesse (so weit möglich) unabhängig voneinander zugewiesen und allen anderen Prozessen die Notwendigkeit einer Synchronisation auf die unabhängigen Prozesse auferlegt. Selbstverständlich müssen nicht alle nicht-repetitiven Module für die Bestimmung des Anfangszustandswahrscheinlichkeitsvektors berücksichtigt werden, sondern nur jene, welche verhältnismäßig früh im Signal-Tracking auftauchen, d.h. jene, bei

denen die erforderliche Einschwingzeit größer wäre als die Zeit, die ein das System durchlaufendes Signal bis zu dieser Komponente mindestens benötigt. Da repetitive Prozesse in wiederkehrender Weise bestimmte Prozesse anstoßen, lässt sich die maximal benötigte Einschwingzeit eines von einem repetitiven Prozess dominierten Systems als das doppelte des maximalen zeitlichen Abstandes der betrachteten Ausgangssignale bestimmen. Diese Abschätzung kann jedoch in einigen Fällen eher konservativer Natur sein, wie z.B. wenn die Verzögerungsdauer des Prozesses #1 des Beispieles deutlich geringer ist, als der maximale zeitliche Abstand zweier Ausgangssignale von Prozess #2.

4.4.2 Zugriffskonflikte

Zugriffskonflikte treten überall dort auf, wo sich zwei oder mehr Prozesse eine Ressource teilen. Dies resultiert entweder in dem in Abschnitt 4.4.3 zu diskutierenden Informationsverlust oder in einem Warteverhalten, wobei zweiteres letztendlich nichts anderes als eine spezielle Verzögerungsursache darstellt. Ereignet sich ein solcher Zugriffskonflikt zufällig, lässt er sich als stochastische Verzögerungsquelle modellieren. Handelt es sich jedoch um einen systeminherenten Konflikt (vgl. hierzu auch Abschnitt 6.3.2), dann muss dieser Konflikt explizit als Warteschlange implementiert werden. Unter einem systeminherenten Konflikt wird hierbei eine Kollission verstanden, die sich aufgrund der Systemevolution ergibt oder die durch mehrfaches (i.S.v. gekoppeltem) Auftreten ihren zufälligen Charakter verloren hat. Abbildung 16 zeigt hierfür ein Beispiel.

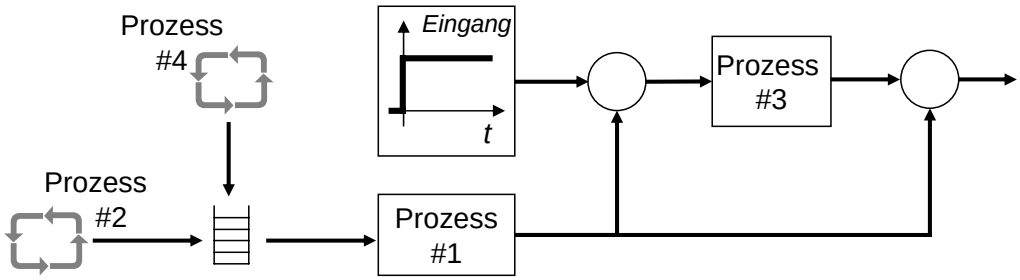

Abbildung 16: Beispiel für die Notwendigkeit, einen Zugriffskonflikt als Warteschlange zu modellieren

Warteschlangen modellieren zu müssen gehört zu den unangenehmsten Dingen in PMC, da es bisher keine Methodik gibt, die hierdurch erzeugte Zustandsraumexplosion zumindest in Grenzen zu halten oder ganz zu vermeiden. Die beiden kritischen Faktoren der Warteschlangenmodellierung sind der notwendige Informationstransport und die Speicherung der (richtigen) Reihenfolge. Der Informationstransport lässt sich unter Verwendung externen Signal-Trackings bis zu einem gewissen Grad auslagern, was zu einer Verringerung der für die Implementierung notwendigen Speicherbedarfsanforderungen führt. Die Modellierung der Reihenfolge lässt sich jedoch nur in engen Grenzen optimieren. Hierzu zählt die Möglichkeit, Nebenprozesse zusammenzufassen, falls es nicht wichtig ist, welcher Nebenprozess einen Warteplatz belegt, sondern lediglich, dass einer der Nebenprozesse diesen Platz belegt hat. Eine weitere Möglichkeit die Zustandsexplosion einzudämmen besteht

darin, eine obere Grenze des Auftretens eines bestimmten Signals innerhalb einer Warteschlange zu finden, d.h. analytisch die maximale Auftrittshäufigkeit herzuleiten oder notfalls durch Annahmen abzuschätzen. Eine Anwendung dieses Punktes ist insbesondere dann sehr vorteilhaft, wenn niemals zwei identische Signale gleichzeitig innerhalb der Warteschlange auftreten oder der Warteprozess eine Logik aufweist, die auftretende Signale zusammenfasst, falls sie informationstechnisch identisch sind.

Die Anfangsbelegung von Warteschlangen analytisch zu bestimmen macht wenig Sinn, obgleich es selbstverständlich möglich ist. Dies liegt daran, dass Komplexität und Nutzen einer solchen Analyse in keinem vertretbaren Verhältnis zueinander stehen. Insbesondere in Systemen der Automatisierungstechnik, in denen alle Prozesse so ausgelegt sein sollten, dass es im Normalbetrieb keinerlei Überlast gibt, ist es einfacher, eine für die Selbstfindung notwendige Einschwingzeit nach den im vorherigen Abschnitt diskutierten (vom Zugriffskonflikt selbst unabhängigen) Faustformeln zu berechnen.

4.4.3 Informationsverlust

In der bisherigen Diskussion wurden lediglich Prozesse berücksichtigt, bei denen ein Signal, unabhängig davon, wie lange es durch einen Prozess verzögert wird, zumindest irgendwann am Ausgang ankommt. Natürlich gibt es auch Prozesse, die nicht terminieren oder zu ihrer Anfangsmarkierung zurückkehren, ohne die zugehörigen Ausgabewerte gesetzt zu haben. In einem solchen Fall würde das Signal-Tracking beliebig lange warten. Um dies zu umgehen, muss jede mögliche Verluststelle in der Analyse berücksichtigt werden. Hierfür kann die notwendige Überwachungsfunktion als Teil des Signal-Trackings implementiert werden. Dieses Vorgehen führt zwar zu einem zusätzlichen Kapazitätenverbrauch, ist aber meist der einzig gangbare Weg und wird im Rahmen des Anwendungsbeispiels (Kapitel 6) explizit dargestellt. Alternativ hierzu ist es oftmals möglich, das den Informationsverlust hervorrufende Modul weitere Informationen erzeugen zu lassen, ein Vorgehen, das sich auf die Einführung eines weiteren Signal-Tracking Moduls abstrahieren lässt.

Der unangenehmste im Zusammenhang mit Informationsverlusten auftretende Effekt ist, dass sich aus der Behandlung desselben meist parallele Warte- und Abfertigungsstrukturen entwickeln, deren PMC-Implementierung sehr viel Speicherplatz benötigt und, da PRISM keine rekursiven Strukturen unterstützt, eine Menge Programmierarbeit mit sich bringt.

In Bezug auf die Bestimmung des Anfangszustands müssen glücklicherweise nur solche informationsverlustbehafteten Prozesse berücksichtigt werden, die sich innerhalb eines kritischen Nebenarms befinden, da in allen anderen Fällen der Informationsverlust nur zu einer Unterbrechung des Signalpfades führt. Letzteres ist aus Sicht der in diesem Abschnitt betrachteten Anfangszustandsbestimmung deshalb unerheblich, da es lediglich das Signal-Tracking, nicht aber den Anfangszustand beeinflusst. Hierzu gehört beispielsweise auch das Auftreten von Eingangssignalen der Typen II und III (vgl. Kapitel 2.2). In den wenigen Ausnahmen, bei denen der Informationsverlust statt keinem ein verfälschtes Ausgabeverhalten zur Folge hat, wird empfohlen, zunächst zu prüfen, ob es möglich ist,

den betroffenen Prozess in einer Weise neu zu modellieren, so dass die Verfälschung des Signals durch ein externes und zufälliges Ereignis hervorgerufen wird. Ist dies nicht der Fall, verbleibt nur, eine Einzelfallbetrachtung durchzuführen.

4.5 Fazit

Bei der Verwendung der PMC ist die richtige Wahl des Initialzustands ein bisher nur für triviale Systeme diskutiertes Problem. Obwohl sich diese Problemstellung auf Grund ihrer Komplexität leider nicht geschlossen angehen lässt, bzw. eine geschlossene Lösung die Analyse des Gesamtsystems vorwegnehmen würde, konnte in diesem Kapitel gezeigt werden, dass es möglich ist, sämtliche in einem NAS auftretenden Systemklassen derart in Verhaltensgruppen aufzuteilen, dass eine praktikable Lösung bestimmbar ist. Je nach Gruppe reichen die möglichen Ansätze von der Ermittlung einer geschlossenen Lösung, wie sie beispielsweise für repetitive Prozesse anzugeben möglich ist, bis zur Implementierung eines kurzen Einschwingvorgangs, wie er z.B. bei der Modellierung von komplexeren Informationsverlusttypen Anwendung findet.

Den einfachsten Fall bilden die repetitiven Prozesse, welche geschlossen und separat behandelbar sind, bzw. – falls Kopplungen mit anderen repetitiven Prozessen vorhanden sind – in direkter Abhängigkeit zueinander diskutiert werden. Im Falle der nicht-repetitiven Prozesse ist es entscheidend, ob diese rein ablaufender Natur sind, d.h., auf ein externes Startsignal warten um anschließend ein bestimmtes Verhalten auszuführen, oder ein in sich erneut abhängiges Verhalten aufweisen. Darüber hinaus ist zu unterscheiden, ob ein solch ablaufender Prozess von einem externen stochastisch unabhängigen Signal, von einem internen aber repetitiven Prozess oder von einem nicht-repetitiven Prozess angestoßen wird. Der letztgenannte Fall lässt sich erneut in bekannter Weise untergliedern. Besondere Aufmerksamkeit ist gefordert, wenn sich im System vorhandene Fehlerquellen oder Zugriffskonflikte auf den Anfangszustand auswirken.

Zum Abschluss dieses Kapitels wird gezeigt, wie die Implementierung des Anfangszustands repetitiver Prozesse in PRISM umgesetzt werden kann.

4.6 Implementierungsmöglichkeiten in PRISM

Die Implementierung in PRISM basiert auf der Verwendung unabhängiger Module. Variablen sollten lokal definiert werden und sind somit auch nur lokal, d.h. vom sie definiert habenden Modul, veränderbar. In PRISM sind alle Variablen global sichtbar, was bedeutet, dass lokale Variablen nicht exportiert werden müssen, aber auch nicht versteckt werden können. Jeder Variablen kann ein Anfangswert zugewiesen werden. Wird hiervon abgesehen, verwendet PRISM automatisch den kleinsten zulässigen Variablenwert. Darüber hinaus bietet PRISM eine `init` Funktion, mittels welcher es möglich ist, auch Anfangswerte komplexerer Struktur zuzuweisen, wie z.B. `a=b` oder `a<5`. Bei Verwendung der `init` Funktion

reagiert der Compiler jedoch leider mit dem folgenden Warnhinweis im Log-File: „*Warning: There are multiple initial states; the result of model checking is for the first one: 0:(0,0)*“.

Die Bedeutung dieser Meldung ist wörtlich zu nehmen: Selbstverständlich ergibt die wiederholte Matrixmultiplikation für jeden Anfangszustand die Wahrscheinlichkeit, einen die gestellte Bedingung erfüllenden Zustand zu erreichen. Auch kann man PRISM mittels verbose option dazu bewegen, all diese Wahrscheinlichkeiten auszugeben. Nur müsste man diese dann von Hand heraussuchen und entsprechend ihrer Auftretenswahrscheinlichkeit wichten, da PRISM die Funktionalität der Bildung des (gewichteten) Mittelwerts bisher leider nicht unterstützt.

Somit verbleibt nur die Möglichkeit, die Anfangswerte unter Verwendung eines Vorprozesses zu ermitteln. Ein solcher Vorprozess wird noch vor dem eigentlichen Systemablauf ausgeführt und weist jeder Variablen entsprechend der gegebenen Wahrscheinlichkeiten einen Anfangswert zu. Folglich muss dieser Vorprozess auch in jedem betroffenen Modul einzeln implementiert werden. Hierfür wurden zwei unterschiedliche Herangehensweisen gefunden, namentlich die Einschritt- sowie die serielle Wertzuweisung.

4.6.1 Einschrittwertzuweisung

Verwendet man diesen Ansatz, wird der Vorprozess auf einen einzigen Automatenschaltvorgang beschränkt. Dies bedeutet, dass ausgehend vom „Null-Zustand“ – dieser wird durch die `init` Funktion zugewiesen und darf im Folgenden nicht wieder angenommen werden – alle Werte innerhalb einer einzigen (auf einmalige Ausführung beschränkten) Anweisungszeile zugewiesen werden:

```
[pre] cyc=0 -> p1:(cyc'=1) + p2:(cyc'=2) + p3:(cyc'=3) + ...;
```

`[pre]` zeigt hierbei an, dass diese Befehlszeile synchron mit den mit `[pre]` beginnenden Befehlszeilen aller anderen Module ausgeführt werden muss. Zur Vermeidung von nichtdeterministischem Verhalten darf pro Modul stets nur eine Codezeile mit erfüllter Bedingung existieren. Da cyc den Wert 0 nur zu Beginn angenommen hat, ist durch `cyc=0` gewährleistet, dass diese Befehlszeile lediglich einmal ausgeführt wird. Alternativ könnte zu diesem Zweck auch eine binäre Variable verwendet werden, womit die zugehörige Befehlszeile folgendermaßen lauten würde:

```
[pre] init -> p1:(cyc'=1)&(init'=false) + p2:(cyc'=2)&(init'=false) + ...;
```

Die erste Version (`cyc=0`) nutzt also einen zusätzlichen Wert der verwendeten Variablen zur Detektion. Dahingegen führt die zweite Version (`init`) eine zusätzliche Variable ein. `cyc=0` ist zu bevorzugen, wenn nur wenige Module diesen Vorprozess ausführen müssen, da die durch den zusätzlichen Variablenwert hervorgerufene Vergrößerung des Zustandsraums durch jenen Faktor gegeben ist, der sich als Summe der Kehrwerte aller betroffenen Variablenmaximalwerte ergibt. Benötigt jedoch eine größere Anzahl an Modulen einen Vorprozess, ist die zweite Version zu bevorzugen, da diese die Größe des Zustandsraums lediglich verdoppelt.

Die restliche Befehlszeile sollte selbsterklärend sein, wobei angemerkt werden muss, dass mit dieser Methodik nur eine unabhängige Variable pro Modul einen Anfangswert zugewiesen bekommen kann, da andernfalls weitere Befehlsschritte notwendig oder die Anfangswerte der beiden Variablen korreliert wären.

Der Vorteil dieser Einschrittwertzuweisungsmethodik liegt in der Tatsache, dass die komplette Wertzuweisung in einem einzigen Schritt abgehandelt werden kann. Darüber hinaus können die auftretenden Anfangswahrscheinlichkeiten – im Gegensatz zur seriellen Wertzuweisung, s.u. – ohne Anpassung direkt implementiert werden. Letzteres spiegelt allerdings den größten Nachteil dieser Methodik wider: Da die Summe aller Wahrscheinlichkeitswerte eins ergeben muss, ist der Wertumfang einer Variablen fest im Programmcode integriert und kann somit nicht durch die Einführung eines Parameters angepasst werden; es sei denn, man würde für jeden möglichen Parameterwert eine eigene Befehlszeile im Programmcode vorsehen. Als Beispiel hierfür diene ein zyklischer, als Ringzähler implementierter Prozess, der die Zykluszeit `Tcyc` aufweisen soll. Für `Tcyc=2` könnte die Vorprozesszeile somit lauten:

```
[pre] cyc=0 & Tcyc=2 -> 1/Tcyc:(cyc'=1) + 1/Tcyc:(cyc'=2);
```

Für `Tcyc=3` müsste die rechte Seite bereits auf drei Elemente erweitert werden:

```
[pre] cyc=0 & Tcyc=3 -> 1/Tcyc:(cyc'=1) + 1/Tcyc:(cyc'=2) + 1/Tcyc:(cyc'=3);
```

Und so weiter für jeden gewünschten Parameterwert `Tcyc`. Um dies zu vermeiden, wurde mit der seriellen Wertzuweisung eine alternative Methodik entwickelt.

4.6.2 Serielle Wertzuweisung

Bei dieser Methodik werden die unterschiedlichen Werte der Variablen einer nach dem anderen zugewiesen (vgl. Abbildung 17). Als Ausgangspunkt diene der erste zuzuweisende Variablenwert. Ist p_1 die Wahrscheinlichkeit, dass dieser erste Wert der Variablen zu Beginn zugewiesen sein soll, kann der Vorprozess mit dieser Wahrscheinlichkeit abgeschlossen werden (die Variable bleibt dann auf diesem Wert „stehen“). Die zugehörige Programmcodezeile lautet insofern:

```
[pre] cyc=1 & initA -> p1:(cyc'=1)&(initA'=false) + (1-p1):(cyc'=2);
```

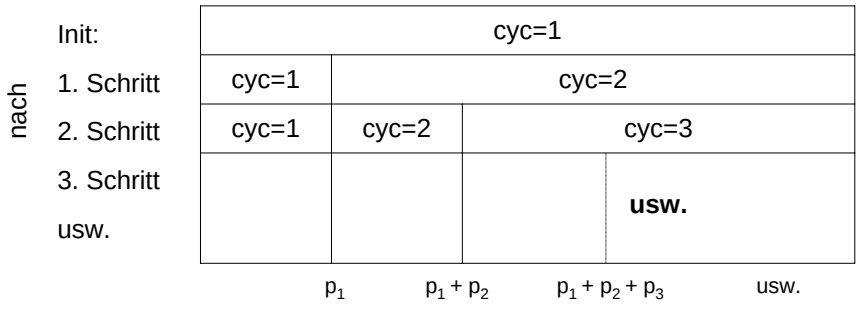

Abbildung 17: Prinzipschema der seriellen Wertzuweisungsmethodik

Nicht so für die mit einer Wahrscheinlichkeit von $1 - p_1$ gegebenen restlichen Fälle, da diese Restmenge ggf. weiter aufgespaltet werden muss. Die Schwierigkeit besteht nun darin, dass für die Implementierung des zweiten Schrittes die durch den ersten Schritt getroffene Aufsplittung berücksichtigt werden muss. Das bedeutet, dass die Wahrscheinlichkeit dafür, dass der Vorprozess nach dem zweiten Schritt beendet ist, nicht $p2$, sondern $q_2 := \frac{p_2}{1-p_1}$ beträgt. Die zweite Vorprozesszeile lautet somit:

```
[pre] cyc=2 & initA -> q2:(cyc'=2)&(initA'=false) + (1-q2):(cyc'=3);
```

Die Wahrscheinlichkeiten der weiteren Codezeilen ergeben sich nach Gleichung 4.3, wobei der zugehörige PRISM-Code analog der Anweisungszeile für q_2 aufgebaut ist.

$$q_1 = p_1, \quad q_2 = \frac{p_2}{1-p_1} \quad \text{und} \quad q_i = p_i \cdot \frac{1 - \sum_{j=1}^{i-2} p_j}{1 - \sum_{j=1}^{i-1} p_j} \text{ für } i > 2 \tag{4.3}$$

Da bei Verwendung dieser Methodik nicht vorhersehbar ist, wieviele Schleifen der Automat tatsächlich ausführen wird, ist es erforderlich, eine Synchronisation mit den Vorprozessen der anderen Module herzustellen, damit alle Module die Vorprozessphase gleichzeitig beenden. Hierfür erhält jedes Modul eine weitere Vorprozessanweisungszeile, welche alle Variablenwerte unverändert lässt (`true`):

```
[pre] !initA & init -> true;
```

Die zugehörige Bedingung ist genau dann erfüllt, wenn der eigentliche Vorprozess dieses Moduls bereits abgeschlossen ist (`!initA`), das überlagerte Kontrollmodul die Vorprozessausführung aber noch nicht beendet hat (`init`). Das überlagerte Kontrollmodul besitzt genau zwei Befehlszeilen und hat lediglich die Aufgabe, die Variable `init` von `true` auf `false` zu schalten und somit das Ende des Vorprozesses zu signalisieren:

```
[pre] initA  |  initB  |  initC  | ... | init  ->  init'=true;
[pre] !initA & !initB & !initC  & ... & init  ->  init'=false;
```

Diese Methodik ermöglicht es in der Tat, für jede beliebige Parameterkonstellation die Anfangswertzuweisung vorzunehmen. Dies ist im Vergleich zur Einschrittwertzuweisungsmethodik ein immenser Vorteil. Allerdings müssen für die gewonnene Universalität mehrere Nachteile in Kauf genommen werden:

- Die Vorprozesse der einzelnen Module haben unterschiedliche Ausführungslänge, auch dann, wenn die Maximalwerte ihrer Variablen identisch sind. Dies liegt daran, dass natürlich jede beliebige Wertekombination eintreten kann und somit Vorprozess A sich z.B. für sein erstes Element entscheidet, während Vorprozess B z.B. das vorletzte oder letzte Element seiner Variablenwertemenge auswählt. Mithin müssen Zwischenschritte und ein diesen Vorprozess überwachendes Modul implementiert werden, was zu einem deutlichen Variablenzuwachs führt.

- Die einzelnen Wahrscheinlichkeitswerte können nicht direkt in den Code übernommen werden, sondern müssen zunächst transformiert werden. Im Ergebnis ist das Nachvollziehen des Programmcodes meist komplexer.
- Die Länge des Vorprozesses entspricht – so lange sie nicht künstlich normiert wurde – immer dem Maximum der jeweiligen Längen der Vorprozesse. Da hierbei sämtliche Kombinationen auftreten können, bedeutet dies, dass auch die Länge des Gesamtvorprozesses variabel ist. Dies bedeutet, dass die Information, innerhalb welcher Zeit ein bestimmter Zustand erreicht worden ist, nicht mehr über das Zählen aller ausgeführten Zustandsübergänge erfolgen kann. Die Anwendung des beschränkten `Until`-Operators würde daher eine nachgeschaltete Normierung erfordern.

Auch für diese Methodik gilt natürlich, dass nicht zwei Variablen ihre Wertzuweisung innerhalb ein und derselben Befehlszeile erhalten können, wenn diese Zuweisungen unabhängig voneinander sein sollen. Im Gegensatz zur Einschrittwertzuweisungsmethodik ist es jedoch problemlos möglich, nach der Zuweisungsprozedur der ersten Variable eine zweite Zuweisung vorzunehmen und erst dann dem Vorprozessüberwachungsmodul die Fertigstellung anzuzeigen.

Anmerkung: Für den Spezialfall, dass alle möglichen Anfangswerte einer Variablen gleichwahrscheinlich sind, ergibt sich unter Verwendung des die Anzahl möglicher Anfangswerte dieser Variablen angebenden Parameters $vMax$ die folgende, Gleichung 4.3 vereinfachende Berechnungsvorschrift:

$$q_i = \frac{1}{vMax - i} \qquad ; 0 \leq i < vMax \tag{4.4}$$

Der zugehörige PRISM-Modulcode ist für diesen Fall in zwei Zeilen angebbar:

```
[pre]  initA & cyc<vMax  -> 1/(vMax-cyc):(initA'=false)
                                + (1-1/(vMax-cyc)):(cyc'=cyc+1);
[pre] !initA & init      -> true;
```

5 Modellierungsansatz

Für die eigentliche Modellierung ist es sinnvoll, zunächst eine Beschreibungssprache zu definieren, welche die strukturbasierte Denkweise des modellierenden Ingenieurs widerspiegelt. Diese, DesLaNAS (Description Language for Networked Automation Systems) benannte Beschreibungssprache ist so gewählt, dass sie einer Modellierungsform entspricht, welche alle möglichen Verhaltensmodi eines NAS beschreiben kann (vgl. Abschnitt 5.1). Das hierbei zugrundeliegende dreigliedrige Modulkonzept wird in Abschnitt 5.2 diskutiert.

Wie Abbildung 18 verdeutlicht, bildet das so gewonnene grafische Modell die Grundlage für einen dreistufigen Transformationsprozess, bei dem zunächst eine Abbildung des in der Beschreibungssprache verfassten grafischen Modells auf eine auf wahrscheinlichkeitsbasierten zeitbewerteten Automaten (probabilistic timed automata, PTA) basierende modulare Automatenstruktur stattfindet. Letztere wird in Abschnitt 5.3 im Detail definiert und diskutiert werden. Das Besondere an dem Umsetzungsschritt von DesLaNAS auf den kontinuierlichen Automaten liegt darin, dass hierbei die Möglichkeit besteht, gewisse Verhaltensmodi als Signalbeobachtungsmodule zu implementieren.

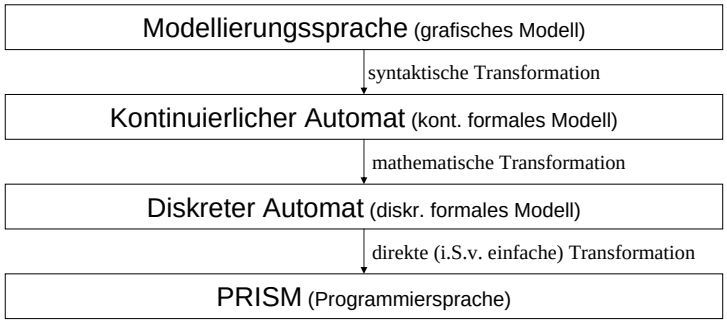

Abbildung 18: Entwurfs- und Transformationsprozess

Bildet man ein kontinuierliches Verhalten in den diskreten Raum ab, um es dort zu analysieren, entstehen eine Reihe von zu berücksichtigenden Artefakten, wie der Herausforderung, eine angemessene Zeitschrittweite zu finden. Das Hauptproblem hierbei ist, dass die Struktur des entstehenden diskreten Modells von der jeweiligen Zeitschrittweite abhängig ist. Je nach Fragestellung benötigt man jedoch feinere oder weniger feine Strukturen, wobei die zeitliche Granularität entscheidenden Einfluss auf den Ressourcenverbrauch hat. In der Folge bedeutet dies, dass sich eine diskrete Modellrepräsentanz nicht für den Entwurf eignet, wohl aber für die eigentliche Berechnung benötigt wird. Daher ist es erforderlich, das Systemmodell in einer zeitkontinuierlichen Version zu speichern und je nach Anwendungsfall das zugehörige diskrete Modell zu erzeugen.

Dieser Vorgang enthält, neben der reinen Überführung auf einen diskreten Automaten (vgl. Abschnitt 5.4), auch die Möglichkeit zur Modellreduktion: Während im Rahmen des kontinuierlichen Modells noch alle systembestimmenden Komponenten modelliert werden, können bei der Diskretisierung all jene Komponenten weggelassen bzw. entsprechend

ersetzt oder abstrahiert werden, deren Eigenverhalten bei der gewählten Schrittweite keinen oder nur noch einen geringen Einfluss auf das Gesamtsystem hat (vgl. Abschnitt 5.5). Hierfür jedoch ist ein großes Maß an Erfahrung notwendig, weshalb bisher noch keine allgemeingültigen Regeln aufgestellt werden konnten.

Die diskrete Beschreibung wurde so gewählt, dass sich die Erzeugung des für die Durchführung der PMC mittels des Verifikationswerkzeuges PRISM erforderlichen Programmcodes als einfache Umsetzung darstellen lässt (Abschnitt 5.6). Abschnitt 5.7 stellt zum Abschluss dieses Kapitels die Grundidee einer Multischrittweitensteuerungsimplementierung vor.

5.1 Beschreibungssprache DesLaNAS

DesLaNAS ist eine grafische Beschreibungsform für PTA, die aufgrund ihrer Strukturierungsformalien PTA modularisiert und infolge der Regeln für die Zustandsverknüpfung PTA auf eine Untermenge einschränkt, welche zum einen zur Beschreibung von NAS ausreicht und zum anderen für die Umsetzung in Analysetools (Diskretisierbarkeit, Determinismus) geeignet ist. Tabelle 3 zeigt zusammenfassend die Anforderungen, welchen eine Beschreibungssprache für NAS genügen sollte.

Tabelle 3: Anforderungen an eine Beschreibungssprache für NAS

Darstellung	Modular	Einzelne Komponententypen kommen mehrfach vor. Hieraus folgt, dass die einzelnen Module parametrierbar sein müssen.
	Ingenieursintuitiv	Auch ohne Kenntnisse in PMC soll der Ingenieur Modelle erstellen können.
	Grafisch	Dies vereinfacht die Modellierung.
Eigenschaften	Keine Nebenläufigkeit	Es ist immer nur ein Gesamtzustand aktiv. Allerdings hat jedes Modul seinen vom restlichen System mehr oder minder unabhängigen Teilprozess.
	Determinismus	Automatisierungssysteme sollten stets deterministisch sein.
	Zeitkontinuität	Modelle sind über kontinuierlicher Zeit beschrieben (es gibt nur eine globale Zeit).
	Diskretisierbarkeit	Für die Nutzung in PMC sind diskrete Modelle erforderlich.
Modellierungsmöglichkeiten	Abbildung von	- Zeitabläufen beliebiger Natur - Abhängigkeiten zwischen den Modulen - statistisch ermitteltem Verhalten - mittels stochastischer Elemente abstrahierter Vorgänge, wie z.B. die Zeitverteilung eines bekannten Prozesses
	Formulierung von	- Eingangssignalen - Abläufen - Anfangsbedingungen - zu verifizierendem Verhalten

DesLaNAS stellt somit eine universelle, von der konkreten Softwareplattform und Analysemethodik unabhängige Beschreibungssprache für NAS dar, mittels derer alle Verhaltensmodi eines NAS abgebildet werden können. Neben den reinen Strukturblöcken sind hierbei insbesondere die Verwendung von Signal-Tracking Modulen zur Erfassung von Abläufen und Eingangssignalen als auch die Parametrierbarkeit zu nennen.

Die in einem NAS auftretenden Zustandsübergänge lassen sich ganz allgemein auf die in Abbildung 19 gezeigten vier Grundtypen abstrahieren:

(1) *Deterministische Verzögerung*: Hierunter versteht man eine fest vorgegebene Zeitspanne, nach deren Ablauf der Übergang umgehend ermöglicht und sofort ausgeführt wird. Beispielhaft sei eine gleichbleibende Bearbeitungszeit oder die maximal zulässige Betriebszeit genannt. Pro Zustand ist lediglich eine deterministische Verzögerung zugelassen, da bei mehreren mit einer deterministischen Verzögerungszeit ausgestatteten Übergängen, stets nur jener Übergang mit der kleinsten Zeitbedingung ermöglicht würde (und das Auftreten zweier identischer Zeitbedingung der Definition eines deterministischen Verhaltens widerspricht).

(2) *Stochastische Verzögerungen* werden mit Hilfe von Wahrscheinlichkeitsdichtefunktionen $d(x)$ beschrieben. Das Integral über $d(x)$ von $x = 0$ bis $x = t$ repräsentiert hierbei die Wahrscheinlichkeit dafür, dass der Übergang innerhalb dieser Zeitspanne ermöglicht und ausgeführt wird. Diese Form des Überganges findet sich überall dort, wo mehrere Nutzer eine Komponente teilen bzw. das genaue (zwar i.d.R. deterministische, dafür aber komplexe) Verhalten einer Komponente mittels einer Übergangsfunktion abstrahiert wird. Beispielhaft sei hierbei die Übertragungszeit durch ein Netzwerk genannt. Wie bereits im Fall deterministischer Verzögerungen ist auch für den stochastischen Fall maximal ein abgehender Übergang (dieses Typs) pro Zustand erlaubt, da andernfalls zwei Übertragungsfunktionen gleichzeitig aktiv sein müssten

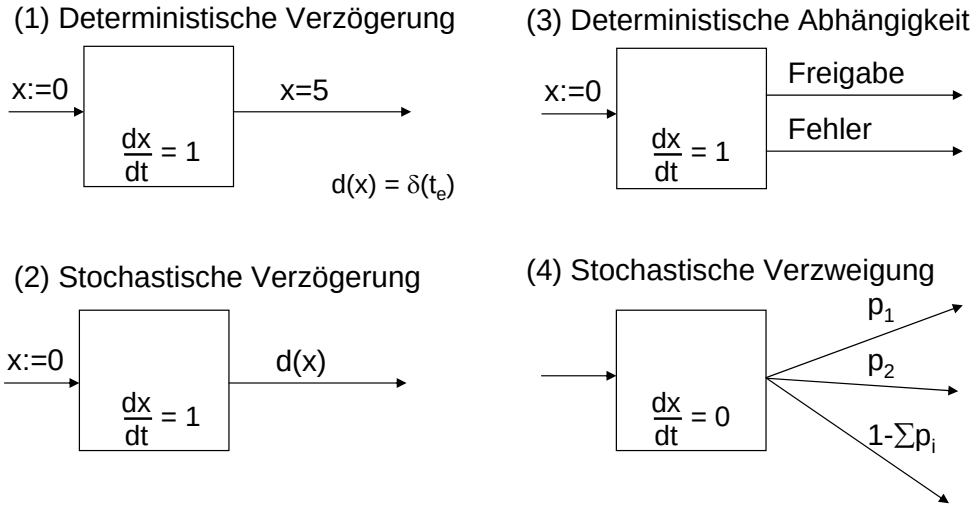

Abbildung 19: Grundtypen der in einem NAS auftretenden Zustandsübergänge

oder eine der beiden nie zum Zuge käme. Das Auftreten zweier zur selben Zeit aktiven Übergangsfunktionen würde bedeuten, dass das Integral über die Dichtefunktionen aller abgehenden Zustandsübergänge nicht mehr eindeutig bestimmt bzw. die Konvergenz zu eins nicht mehr gewährleistet werden kann.

Anmerkung: Jede deterministische Verzögerung $x = t_e$ kann als $d(x) = \delta(t_e)$ geschrieben werden, worin t_e den Aktivierungszeitpunkt und $\delta(t_e)$ eine Kurzschreibweise für den Diracschen Deltaimpuls an der Stelle $x = t_e$ repräsentieren. Da jedoch eine solche Schreibweise im Bedingungsformulierungsschritt etwas ungewohnt ist, wird diese Umwandlung in eine, eine stochastische Verzögerung repräsentierende, Dichtefunktion durch den Transformationsprozess aus Abbildung 18 automatisch durchgeführt.

(3) *Deterministische Abhängigkeit*: In diesem Fall ist die Ermöglichung des Übergangs an den Eintritt einer vorgegebenen Situation gekoppelt. Wird eine Bedingung angegeben, so ist hiermit das beim Auslösen des zugehörigen Übergangs erzeugte Ereignis gemeint. Im Gegensatz zu den vorherigen beiden Zustandsübergangsgrundtypen kann ein Zustand mehrere abgehende Übergänge dieses Typs aufweisen, wobei allerdings aufgrund der Ereignischarakteristik niemals zwei Bedingungen zur selben Zeit erfüllt sein können. Hieraus folgt direkt, dass UND-Verknüpfungen innerhalb von Bedingungen nicht möglich sind (siehe unten). Dieser Übergangstyp tritt in einem NAS z.B. beim Schreiben/Lesen der SPS oder beim Senden einer Anfrage durch die SPS-I/O auf.

(4) *Stochastische Verzweigung*: Im Gegensatz zu den vorherigen Übergangstypen erfolgt die Ausführung einer stochastischen Verzweigung unverzüglich, wobei jedoch die Wahl des Pfades stochastischer Natur ist (vgl. hierzu auch Gleichung 5.12). Hieraus folgt,

1. dass eine stochastische Verzweigung nur mit mindestens zwei abgehenden Übergängen Sinn macht,
2. dass die Summe der Eintrittswahrscheinlichkeiten aller abgehenden Übergänge eines Zustandes eins ergeben muss und
3. dass der zugehörige Zustand nur transienter[2] Natur ist, was bedeutet, dass während dem Aufenthalt in diesem Zustand keinerlei Zeit vergeht ($\frac{dx}{dt} = 0$).

Zusammengenommen bedeutet dies, dass eine stochastische Verzweigung unter Weglassung des transienten Zustandes in Form eines gewichteten Verzweigungsvektors an jede der anderen drei Übergangsgrundtypen direkt (i.S.v. seriell) hinzugefügt werden kann. Stochastische Verzweigungen treten in NAS-Strukturen besonders im Zusammenhang mit Fehlern auf, wie z.B., dass die Wahrscheinlichkeit einer Fehlübertragung oder eines Messfehlers bei einem Prozent liegt.

[2]Lunze verwendet den Begriff des *transienten* Zustands für ein anderes Verhalten: In [Lunze, 2006] wird ein Zustand als transient bezeichnet, wenn es keinen Pfad gibt, der von diesem Zustand wieder in diesen Zustand zurückführt. Einen Zustand, von dem aus alle abgehenden Pfade denselben mit Sicherheit erneut passieren werden, bezeichnet Lunze als *rekurrent*. Zustände, deren Verweildauer immer gleich Null ist, werden in [Lunze, 2006] weder thematisiert noch benannt.

Während selbstverständlich alle vier Grundtypen in beliebiger Reihenfolge und jeweils durch einen weiteren Zustand getrennt hintereinander angeordnet werden können, ist die Verwendung von unterschiedlichen Übergangsgrundtypen am selben Zustand nur mit Einschränkungen erlaubt, bzw. im Falle einer Kombination aus (1) und (2) sogar verboten. Da sich (1) in (2) überführen lässt, folgt dies direkt aus dem Ausschluss zweier zur selben Zeit aktiven Dichtefunktionen. Eine Verwendung von Grundtyp (4) mit einer der anderen Grundtypen, ausgehend vom selben Zustand, ist, aufgrund der sofortigen Aktivierung dieses Übergangstyps, selbstredend nur in der zuvor erläuterten Form des Anhängens eines transienten Zustandes möglich und sinnvoll. Somit verbleibt die Diskussion einer Verknüpfung des Grundtyps (3) mit (2) bzw. (1). Hierbei lassen sich drei Varianten unterscheiden:

(A) Verknüpfung zweier Aktivierungsbedingungen innerhalb eines Übergangs, wie beispielhaft im linken Automaten von Abbildung 20 gezeigt. Hierbei wird das Verhalten einer Maschine modelliert, welche im Bereitschaftszustand (`Bereit`) wartet, bis z.B. ein Werkstück eintrifft oder ein ‚Start'-Knopf gedrückt wird. Die Bearbeitung dauere dann 5 ms (deterministische Verzögerung) und kann durch die Aktivierung des ‚Stopp'-Knopfes abgebrochen werden (deterministische Abhängigkeit). Die zugehörige Übergangsbedingung weist daher eine ODER-Verknüpfung (Disjunktion) der beiden Bedingungen ‚x=5' und ‚Stopp' auf. Da es sich bei den zu verknüpfenden Bedingungen um Ereignisse handelt, die definitionsgemäß nie zum gleichen Zeitpunkt aktiviert werden können (andernfalls würde es sich um ein und dasselbe Ereignis handeln), ist als Verknüpfungsoperator lediglich der ODER-, nicht aber der UND-Operator zulässig. Die UND-Verknüpfung muss daher unter Zuhilfenahme eines Zwischenzustandes realisiert werden (erst das erste und dann das zweite Ereignis sowie erst das zweite und dann das erste Ereignis).

(B) Verwendung zweier unterschiedlicher Übergangstypen ausgehend von ein und demselben Zustand, wie beispielhaft im rechten Automaten von Abbildung 20 gezeigt. Hierbei ist das Verhalten eines Prozessors dargestellt, welcher als ersten Schritt sämtliche Inputs (Eingänge) einliest, diese anschließend verarbeitet und die zugehörigen Ergebnisse abschließend ausgibt. Der Einlesevorgang daure t_r Millisekunden, die Bearbeitung 5 Millisekunden und für die Ausgabe werden t_w Millisekunden veranschlagt.

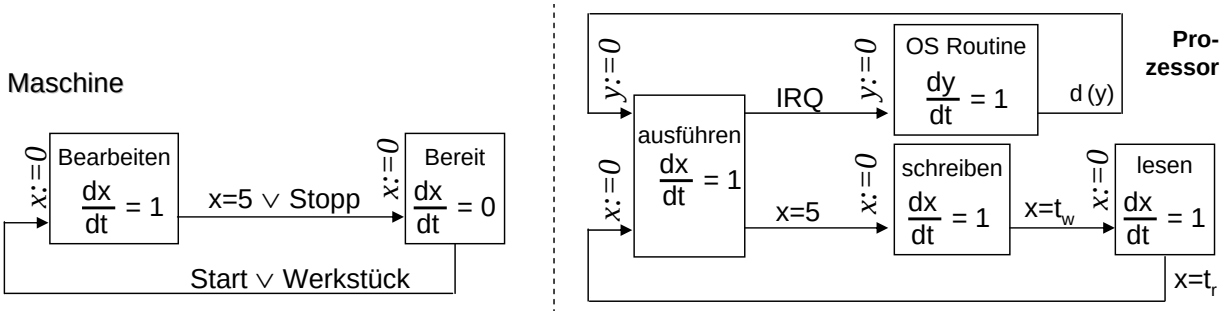

Abbildung 20: Beispiele für Zustände mit gemischten Übergangstypen

Neu ist hierbei das Verhalten im `Bearbeiten`-Zustand. Während der Bearbeitung kann nämlich ein Interrupt IRQ des Betriebssystems eintreffen, welcher die Bearbeitung unterbricht und die extern angeforderte Betriebssystemroutine startet. Während die Betriebssystemroutine abgearbeitet wird – die hierfür erforderliche Zeit wird durch eine Wahrscheinlichkeitsverteilung über der Routinenzeit y beschrieben – ist die Bearbeitung des Hauptprogramms gestoppt, d.h., die zugehörige Bearbeitungszeit x erfährt keine Veränderung bis die Betriebssystemroutine abgearbeitet und der Automat folglich in den `Bearbeiten`-Zustand zurückgekehrt ist.

Anmerkung: In der Literatur wird der in diesem Beispiel beschriebene Unterprozess als „Stopwatch"-Automat referenziert [Cassez und Larsen, 2000], wobei die Idee von Uhren, die zwischenzeitlich angehalten werden dürfen, bereits in [Čerāns, 1992] thematisiert wurde.

Allgemein lässt sich festhalten, dass es möglich ist, eine beliebige Anzahl Übergänge des Grundtyps „deterministische Bedingung" mit maximal einem mit einer Zeitbedingung ausgestatteten Übergang von einem gemeinsamen Zustand parallel ausgehend zu verwenden.

(C) Die Überlagerung von (A) und (B). Hierbei ist zu beachten, dass es mit der hier beschriebenen kontinuierlichen Automatenstruktur nicht möglich ist, eine Auswahl zwischen mehreren mit (zeitlichen!) Dichtefunktionen behafteten Übergängen aufgrund der Erfülltheit einer zugehörigen deterministischen Abhängigkeit zu treffen, da dies erstens einer UND-Verknüpfung entsprechen und zweitens gegen den Grundsatz der Nichtzulässigkeit mehrerer Zeitbedingungen verstoßen würde.

Zu den Einschränkungen gehört ebenfalls, dass der hier vorgeschlagene Automat die Trennung von Zeitbereichen und Ereignissen erzwingt. Eine Bedingung der Form Stopp $\wedge\ x \geq 3$ muss daher in zwei Schritte zerlegt werden: Einen ersten Zustand, in dem der Automat verbleibt, bis $x = 3$ erfüllt ist und einen direkt darauf folgenden, in dem der Automat abwartet, bis Stopp eintritt. Eine Bedingung der Form Stopp $\wedge\ x < 3$ lässt sich hingegen sehr elegant an einem mit zwei abgehenden Übergängen (Stopp sowie $x = 3$) ausgestatteten Zustand realisieren. Grundsätzlich nicht zugelassen sind reine Zeitbedingungen der Form $x \diamond 3$ mit $\diamond \in \{<, \leq, \geq, >\}$, welche lediglich einen Zeitrahmen vorgeben, ohne die Wahrscheinlichkeitsdichte innerhalb dieses Zeitrahmens anzugeben.

Zusammenfassend ergibt sich der in Abbildung 21 dargestellte verallgemeinerte Übergangstyp. Dieser kann einerseits (I) eine Zustandsbedingung, eine zeitliche Bedingung oder eine Disjunktion aus ein oder mehreren Zustandsbedingungen und maximal einer Zeitverteilung aufweisen. Andererseits (II) kann jedem Übergang ein Wahrscheinlichkeitsvektor, der auf eine Menge von Nachfolgezuständen abbildet, zugeordnet werden. Die Verwendung mehrerer verallgemeinerter Übergangstypen, ausgehend von ein und demselben Zustand, unterliegt hierbei den zuvor erläuterten Einschränkungen. Darüber hinaus muss, um die Funktionsfähigkeit des entstehenden Automaten zu garantieren, jeder Zustand mindestens einen Übergang besitzen, dessen Bedingung in endlicher Zeit erfüllt worden sein wird (Verklemmungsfreiheit).

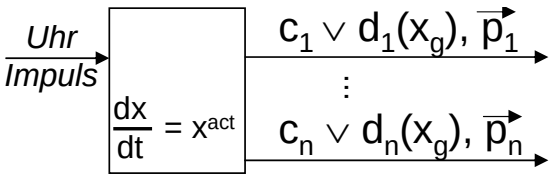

Abbildung 21: Verallgemeinerter Zustandsübergang mit deterministischen Abhängigkeiten c_i, deterministischer oder stochastischer Verzögerung $d_i(x_g)$ und stochastischen Verzweigungen $\vec{p_i}$

5.2 Klassifizierung

Modularer Ansatz

Die Verwendung eines modularen Ansatzes hat vielfältige Vorteile, wobei insbesondere die hierdurch deutlich verringerte Komplexität des Systementwurfs und die Wiederverwertbarkeit von bereits modellierten Systemkomponenten zu benennen sind. Beides reduziert sowohl die Fehleranfälligkeit als auch den für die Modellierung erforderlichen Zeitaufwand. Ebenfalls für eine modulare Modellierung spricht, dass in einem NAS viele Komponenten mehrfach auftreten (z.B. Switche, I/O-Karten, SPSen). Mit der Modularisierung des Gesamtsystems geht jedoch auch die Notwendigkeit einer systemweiten Zeitsynchronisation einher.

Das interne Verhalten der Module wird durch die in Abschnitt 5.3 vorzustellende, an die speziellen Bedürfnisse von NAS angepasste Form von wahrscheinlichkeitsbasierten zeitbewerteten Automaten (PTA) repräsentiert. Von außen betrachtet handelt es sich bei den Modulblöcken um E/A-Automaten [Lunze, 2006]. Den Einzelautomaten ist es einerseits gestattet, Zustandsübergänge asynchron auszuführen und andererseits Ein-/Ausgangsvariablen aufzuweisen (also Ereignisse aufzunehmen, bzw. zu generieren). Die zugrunde liegende Modellierung ist somit funktionsblockorientiert. Das Gesamtsystem ergibt sich schlussendlich durch parallele Komposition aller Teil-/Einzelautomaten; der so entstandene Automat gibt also das komplette Verhalten der zu modellierenden Anlage wieder.

Eigenschaftsauswahl

Der oftmals schwierigste Schritt ist die Bestimmung der zu prüfenden Eigenschaften und die damit verbundene Implementierung des zugehörigen Signalbeobachtungsprozesses. Da mittels PMC lediglich Wahrscheinlichkeiten oder Wahrheitswerte ermittelt werden können, muss jede Fragestellung auf die Form „mit welcher Wahrscheinlichkeit ist ...“ oder „ist die Wahrscheinlichkeit für ... kleiner/größer als ...“ gebracht werden. Aus dem Interesse nach einer Zeitverteilung wird also die Frage, wie groß die Wahrscheinlichkeit dafür ist, dass sich die gesuchte Zeit in einem vorzugebenden Intervall befindet.

Wie bereits in Kapitel 2.2 postuliert, lassen sich Eigenschaften von ereignisdiskreten Systemen nicht generalisieren, sondern lediglich auf eine Menge von Templates abbilden, welche gewisse Eigenschaften in generischer Formulierung widerspiegeln. Die Erarbeitung eines

solchen Templatessatzes für NAS könnte daher ebenso Thema einer auf dieser Arbeit aufbauenden Dissertation sein, wie die Untersuchung der Frage, in wie weit sich die so generierten Templates automatisiert in Signalbeobachtungsmodule (vgl. Abschnitt 4.3) umsetzen lassen. Die Schwierigkeit hierbei liegt in der Notwendigkeit einer Systemabstraktion, welche es ermöglicht, Eigenschaften (wie z.B. die der Antwortzeitverteilung) über einen derartigen Templatessatz allgemein zu definieren und den sich daraus ergebenden Ablauf dann strukturabhängig herunterzubrechen.

Modultypen

Für den Entwurf eines Modells zur Verifikation mittels wahrscheinlichkeitsbasierter Modellverifikation wird ein dreigliedriges Vorgehen vorgeschlagen.

1. Zunächst werden Einzelmodelle der **Grundfunktionen** als unabhängige Module auf Automatenbasis modelliert.
2. Im zweiten Schritt werden diese Module entsprechend ihrer Auftretenshäufigkeit im Modell instanziiert und um **architekturabhängige Verknüpfungen** ergänzt.
3. Im abschließenden dritten Schritt wird das Modell um die für PMC notwendige Signalverfolgung (**Signal-Tracking**) ergänzt und mit gültigen Anfangswerten initialisiert.

Die Gruppe der **Grundfunktionen** zeichnet sich dadurch aus, dass sie unabhängig vom jeweiligen Problem oder den gesuchten Eigenschaften implementiert werden kann. Ein wesentlicher Bestandteil dieser Kategorie wäre z.B. die Grundfunktion einer Anlagen-I/O, den aktuellen Sensorwert auf Anfrage zurückzuliefern, aber auch der periodische Durchlauf einer Speicherprogrammierbaren Steuerung (SPS). Die Menge der Grundfunktionen lässt sich in die in Abschnitt 4.1 definierten repetitiven und nicht-repetitiven Prozesse unterteilen. Obwohl es möglich wäre, eine Modellierung zu wählen, bei der alle Prozesse auf die Verwendung spezifischer Ein- und Ausgangsinformationen (wie z.B. Zeitstempel, Nummerierung, Zieladressen ...) ausgelegt sind, ist es einfacher – und für komplexe Systeme, in Bezug auf Speicher- und Rechenzeitbedarf, deutlich günstiger – alle Prozesse so generisch als möglich zu erzeugen und durch architekturabhängige Verknüpfungen und Signal-Tracking Module zu ergänzen, welche dann die Informationsbeförderung übernehmen. Grundfunktionen lassen sich als fertige Modulbausteine innerhalb der grafischen Benutzeroberfläche hinterlegen. Alle Grundfunktionen müssen entsprechend ihrer Auftretenshäufigkeit im Modell instanziiert werden und bedürfen der problemspezifischen Anpassung.

Im zweiten Schritt werden verknüpfende und problemspezifische Module hinzugefügt. Beispielsweise muss eine instanziierte SPS wissen, welche I/O-Module sie abfragen muss und wie sie – falls dies explizit modelliert werden soll – die empfangenen Werte verarbeiten soll. Ebenso muss hinterlegt werden, welche I/O-Karte an welchen Switch angeschlossen ist und wo ggf. Warteschlangen vorzusehen sind. Da derartige Spezifikationen dadurch gekennzeichnet sind, dass sie die genaue Architektur (i.S.v. Funktionsweise) des Systems

wiedergeben, werden sie als Gruppe der „**Architekturabhängigen Verknüpfungen**“ klassifiziert. Architekturabhängige Verknüpfungen sind zeitunabhängig und rein deterministischer Natur.

Zur Gruppe der Signalverfolgungsmodule (**Signal-Tracking**) gehören alle Module, die die Erfüllung der gesuchten Eigenschaft oder die Abarbeitung eines damit in direktem Zusammenhang stehenden Teilprozesses mitverfolgen (vgl. Abschnitt 4.3).

Im Idealfall ist es möglich, das Systemmodell direkt aus unabhängigen Einzelelementen dieser drei Modulgruppen zusammenzusetzen. Allerdings gilt dies nur für wenige Beispiele und die Vorraussetzung, auf Speicherplatz nicht achten zu müssen. In allen anderen Fällen ist einem, auf Basis dieser drei Bauklassen vorgenommenen, Entwurfsprozess noch eine Reduktion bzw. Verfeinerung nachzuschalten. Dieser Vorgang erfordert jedoch ein breites Spektrum an Erfahrung beim implementierenden Ingenieur, da hierbei nicht nur die Frage beantwortet werden muss, welches Komponentenverhalten in Form welcher Gruppe implementiert wird, sondern insbesondere auch ausreichend Systemkenntnis erforderlich ist, um entscheiden zu können, welche Komponenten weggelassen oder in abstrahierter Form dargestellt werden können.

Beispiel

Als anschauliches Beispiel soll ein aus Sensor, SPS und Stellglied bestehendes (klassisches) Automatisierungssystem betrachtet werden. Die zu untersuchende Fragestellung betrifft die Analyse der Antwortzeitverteilung dieses direkt verdrahteten Systems (im Gegensatz zu den NAS-Strukturen, welche in Kapitel 6 diskutiert werden). Als Antwortzeit sei hierbei die Verzögerung definiert, die sich zwischen einem Signalwechsel an einem Eingang des Systems und der Reaktion am zugeordneten Ausgang ergibt. Dabei wird angenommen, dass die SPS in den untersuchten Szenarien die Reaktion auf einen geänderten Eingangswert innerhalb eines Zyklus berechnet (z.B. O1 := I1). Weiter wird angenommen, dass der neue Signalwert am Eingang des Systems mindestens so lange ansteht, wie es zur Erfassung durch das System erforderlich ist (Signaltyp I). Der zu beobachtende Ablauf beginnt mit dem Eintreffen des externen Signals am Sensor und endet, sobald die SPS den entsprechenden Ausgang aktiviert hat. Da Filterung und andere Effekte vernachlässigt werden (bzw. lediglich durch eine am Ende hinzuaddierte Zeitverzögerung von 1 ms Berücksichtigung finden), lässt sich das Signal-Tracking Modul, wie in Abbildung 22 gezeigt, modellieren. Die eine im System vorhandene SPS wurde hierbei mit SPS1 benannt und das beim Verlassen des Zustandes `schreiben` erzeugte Ereignis konsequenterweise als SPS1.schreiben. Gleiches gilt für das Ereignis SPS1.lesen.

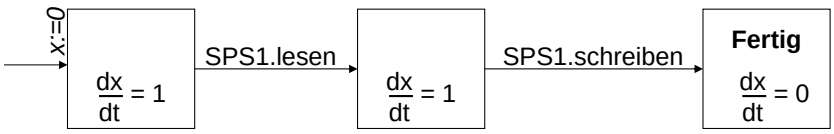

Abbildung 22: Signal-Tracking zur Antwortzeitanalyse einer direkt verdrahteten SPS

Das SPS-DesLaNAS-Modul (vgl. Abbildung 23) hat hierbei die drei Zustände `schreiben`, `lesen` sowie `ausführen`, die beiden Ausgänge *schreiben* und *lesen* und weist zyklisches Verhalten auf. Beim Eintritt in den mit `schreiben` gekennzeichneten Zustand wird die lokale Uhrvariable x zurückgesetzt. In diesem Zustand werden die Ausgänge gesetzt. Nach Ablauf der zugehörigen Zeitdauer ($x = t_w$) wird in den Zustand `lesen` gewechselt. Nachdem das Einlesen der Eingänge begonnen wurde ($x = t_r + t_w$) wechselt der Automat in den mit `ausführen` gekennzeichneten Zustand, um dort zu verweilen, bis sämtliche Befehlszeilen sicher abgearbeitet wurden (modelliert durch eine obere zeitliche Schranke). Selbstverständlich verfügt eine SPS über mehr als diese drei Zustände, wie z.B. die Wartezeit zwischen Befehlsausführung und Ausgabe, welche notwendig zum Erreichen einer gleich bleibenden Zyklusdauer ist. Für die hier durchgeführte Analyse sind jedoch lediglich zwei Ereignisse von Interesse, nämlich: Der Zeitpunkt zu dem die Werte ausgegeben worden sind (Verlassen des Zustandes `schreiben`) und der Zeitpunkt, ab dem die neuen Werte eingelesen werden, sich eine Änderung am Eingang also nicht mehr auf die im nächsten Berechnungsschritt verwendeten Variablen auswirken. Um die Möglichkeit einer detaillierteren Modellierung dieses Moduls ohne Änderung der Schnittstellen offen zu halten, wird der Zeitpunkt des Lesebeginns als Verlassen des Zustands `lesen` modelliert. Diese Ausgangsereignisse werden in der grafischen Darstellung an die Zustandsübergänge geschrieben.

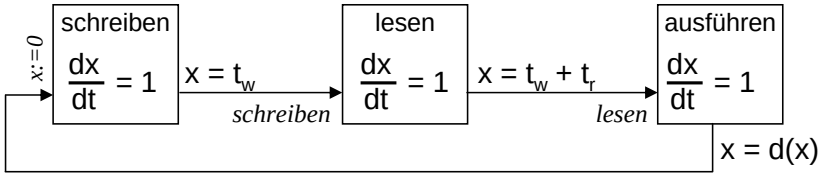

Abbildung 23: SPS-Modul

Die zugehörige Instanziierung und Parametrierung dieser beiden Module (SPS und Signal-Tracking) ist in Abbildung 24 gezeigt. Jeder Modulblock führt hierbei in der Kopfzeile den Namen sowie die Klasse. Darunter finden sich auf der linken Seite die Liste der Parameter, in diesem Fall also t_w, t_r und d(x), und auf der rechten Seite die Liste der Anfangswerte. P^0 steht hierbei für die Wahrscheinlichkeit, dass sich die SPS zum Zeitpunkt 0 im zugehörigen Zustand befindet. $t_{cyc} = 10$ ist die, in diesem Beispiel als konstant angenommene, Zykluszeit. X^0 gibt an, mit welcher Wahrscheinlichkeit die entsprechende Uhr zum Zeitpunkt $t = 0$ einen bestimmten Wert aufweist. GV(a,b) bedeutet gleichverteilt im Intervall (a,b].

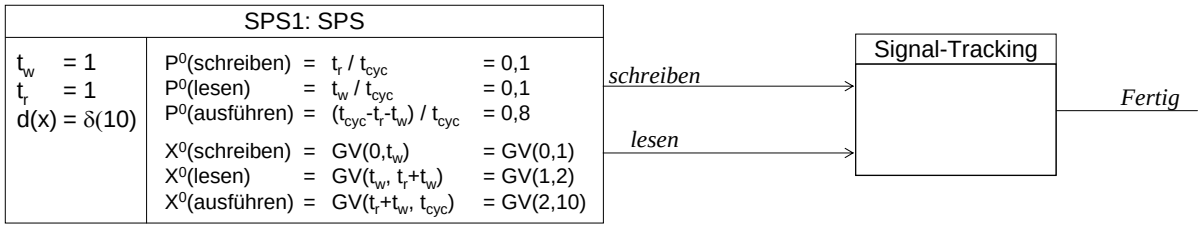

Abbildung 24: Instanziierung und Parametrierung der Antwortzeitanalyse einer direkt verdrahteten SPS

Die Analyse führt zu der in Abbildung 25 dargestellten diskreten Antwortzeitverteilung. Die Plateaubreite entspricht der SPS-Zykluszeit von 10 ms. Der Wert 10% bei 15 ms bedeutet z.B., dass die Wahrscheinlichkeit einer Antwortzeit zwischen 14 und 15 ms 10% beträgt (die Diskretisierung wurde mit einer Zeitschrittweite von 1 ms durchgeführt). Bei den dargestellten Punkten handelt es sich somit um relative Häufigkeiten. Da Balkendiagramme jedoch – insbesondere für komplexere Beispiele – relativ schwer lesbar sind, wurde entschieden, die diskreten Punkte durch gerade Linien zu verbinden.

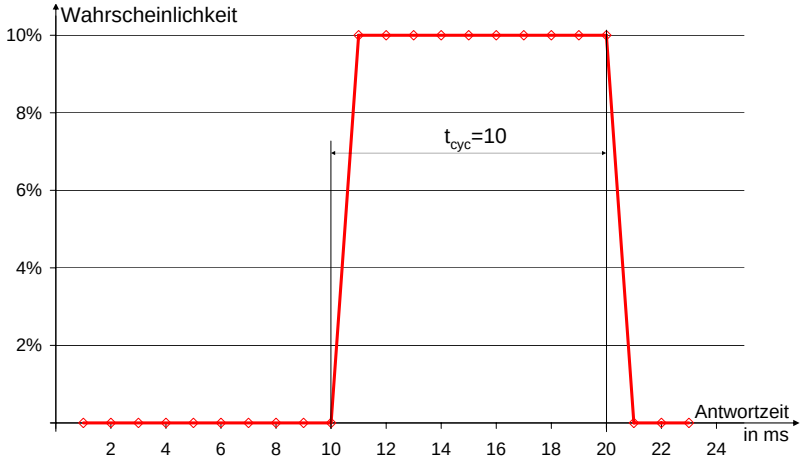

Abbildung 25: Antwortzeitverhalten einer direkt verdrahteten SPS

5.3 Zeitkontinuierlicher Automat

Die korrekte Modellierung des mit physikalischen Vorgängen gekoppelten Automatisierungssystems erfordert die Berücksichtigung der Kontinuität der zugrunde liegenden Prozesse und somit die Verwendung einer kontinuierlichen Zeitbasis zur Beschreibung der Eintrittszeitpunkte von Zustandsänderungen. Dies berücksichtigend schlugen [Alur und Dill, 1994] die Verwendung von zeitbewerteten Automaten (Timed Automata, TA) vor. Dabei verwendeten sie eine dichte (engl.: dense) Zeitskalierung, bei der Zeiten durch rationale Zahlen dargestellt werden, wobei eine künstliche Genauigkeitsschranke festlegt, ab wann zwei Zahlen als gleich zu behandeln sind. Bei der Verwendung dieses Ansatzes ist es immer noch notwendig unterschiedliche Zeiten zu integrieren um diesselbe Charakteristik zu erhalten. Dies wird durch die Verwendung von Regionengraphen (region graphs [Henzinger et al., 1992]) und Uhrregionen (clock regions [Alur et al., 1993; Alur und Dill, 1994]) bewerkstelligt. Die zur Modellierung stochastischer Verteilungen und bedingter Eintrittswahrscheinlichkeiten erforderliche Erweiterung führt auf die wahrscheinlichkeitsbasierten zeitbewerteten Automaten (probabilisitic timed automata, PTA, [Alur et al., 1991]).

Wie in Abbildung 18 gezeigt, wird die in DesLaNAS erstellte Modellrepräsentation in eine kontinuierliche Automatendarstellung überführt und somit formalisiert. Diese, auf PTA-Basis definierte Darstellung soll für die in NAS existenten Strukturen optimiert und so gewählt sein, dass sich die in Abschnitt 5.4 zu definierende, zugehörige diskrete

Automatendarstellung direkt nach PRISM umsetzen lässt. Der Hauptunterschied zwischen dieser, in diesem Abschnitt vorzustellenden und anderen PTA-Definitionen (z.B. [Alur et al., 1991; Jensen, 1996; Bérard et al., 2001; Beauquier, 2003; Kwiatkowska et al., 2007]) liegt darin, dass auch Uhren Anfangsverteilungen aufweisen. Hierdurch wird der in Kapitel 3 erläuterten Zeittransformation Rechnung getragen, die den Systemstart „über“ dem laufenden System vorsieht. Darüber hinaus wurden die im Abschnitt 5.1 erläuterten Restriktionen aufgenommen. Dennoch ist der hier definierte kontinuierliche Automat mächtiger, als dies für die Umsetzung der vorgestellten Beschreibungssprache DesLaNAS notwendig gewesen wäre.

Der für die Beschreibung von NAS ausgelegte zeitkontinuierliche Automat ist durch die folgende Notation gegeben

$$\mathtt{A} = (\mathtt{Z}, \mathtt{Pr}^0, \mathtt{X}, \mathtt{X}^0, \mathtt{X}^{\mathtt{act}}, \mathtt{D}, \mathtt{L}, \mathtt{X}^{\mathtt{R}}) \tag{5.1}$$

und als parallele Komposition über die durch

$$\mathtt{A}_\mathtt{i} = (\mathtt{Z}_\mathtt{i}, \mathtt{Pr}^0_\mathtt{i}, \mathtt{X}_\mathtt{i}, \mathtt{X}^0_\mathtt{i}, \mathtt{X}^{\mathtt{act}}_\mathtt{i}, \mathtt{W}_\mathtt{i}, \mathtt{V}_\mathtt{i}, \mathtt{C}_\mathtt{i}, \mathtt{D}_\mathtt{i}, \mathtt{Y}_\mathtt{i}, \mathtt{L}_\mathtt{i}, \mathtt{X}^{\mathtt{R}}_\mathtt{i}) \tag{5.2}$$

gegebenen Teilautomaten $\mathtt{A}_\mathtt{i}$ ($\mathtt{i} = 1, \ldots \mathtt{n}$), definiert. Die einzelnen Elemente haben die im Folgenden aufgeführte Bedeutung, wobei die Indizes $\mathtt{i}$ für den i-ten Teilautomaten, $\mathtt{i}_\mathtt{g}$ für die g-te Uhr (des i-ten Teilautomaten), $\mathtt{i}_\mathtt{k}$ für den k-ten Zustand, $\mathtt{i}_{\mathtt{k}_\mathtt{j}}$ für den j-ten vom k-ten Zustand abgehenden Übergang und $\mathtt{i}_{\mathtt{k}_{\mathtt{j}\mathtt{h}}}$ für den h-ten stochastischen Verzweigungsast des $\mathtt{i}_{\mathtt{k}_\mathtt{j}}$-ten Übergangs stehen (vgl. Abbildung 26).

$\mathtt{Z}_\mathtt{i}$ stellt die Menge der Zustände des i-ten Teilautomaten dar. Die Mächtigkeit $|\mathtt{Z}_\mathtt{i}|$ der Menge $\mathtt{Z}_\mathtt{i}$ wird als endlich angenommen.

$\mathtt{Pr}^0_\mathtt{i}$ ordnet jedem Zustand $\mathtt{z}_{\mathtt{i}_\mathtt{k}} \in \mathtt{Z}_\mathtt{i}$ jene Wahrscheinlichkeit zu, mit der der Automat $\mathtt{Z}_\mathtt{i}$ im Zustand $\mathtt{z}_{\mathtt{i}_\mathtt{k}}$ startet. Dabei gilt: $\sum_\mathtt{k} \mathtt{Pr}^0_\mathtt{i}(\mathtt{z}_{\mathtt{i}_\mathtt{k}}) = 1$.

$\mathtt{X}_\mathtt{i}$ steht für die Menge der Uhren, $\mathtt{X}_\mathtt{i} = \{\mathtt{x}_{\mathtt{i}_1}, \mathtt{x}_{\mathtt{i}_2}, \ldots, \mathtt{x}_{\mathtt{i}_{\mathtt{n}_\mathtt{imax}}}\}$, $\mathtt{n}_\mathtt{imax}$ ist dabei die Anzahl der Uhren in $\mathtt{X}_\mathtt{i}$.

$\mathtt{X}^0_\mathtt{i}$: $\mathtt{Z}_\mathtt{i} \times \mathbb{R}^+_0 \to \mathbb{R}^+_0$ ordnet jeder Uhr $\mathtt{x}_{\mathtt{i}_\mathtt{g}} \in \mathtt{X}_\mathtt{i}$ für jeden Zustand $\mathtt{z}_{\mathtt{i}_\mathtt{k}} \in \mathtt{Z}_\mathtt{i}$ einen stochastisch gesetzten Anfangswert zu. $\mathtt{X}^0_\mathtt{i}(\mathtt{z}_{\mathtt{i}_\mathtt{k}}, \mathtt{x}_{\mathtt{i}_\mathtt{g}})$ wird dabei als Dichtefunktion angegeben.

$\mathtt{X}^{\mathtt{act}}_\mathtt{i}$: $\mathtt{Z}_\mathtt{i} \times \mathtt{X}_\mathtt{i} \to \{0, 1\}$ gibt an, ob die Uhr $\mathtt{x}_{\mathtt{i}_\mathtt{g}} \in \mathtt{X}_\mathtt{i}$ im Zustand $\mathtt{z}_{\mathtt{i}_\mathtt{k}}$ aktiv oder inaktiv ist:

$$\mathtt{X}^{\mathtt{act}}_\mathtt{i}(\mathtt{z}_{\mathtt{i}_\mathtt{k}}, \mathtt{x}_{\mathtt{i}_\mathtt{g}}) = \begin{cases} 0 & \text{falls inaktiv} \\ 1 & \text{falls aktiv} \end{cases}$$

Uhren, die nicht aktiviert sind, behalten ihren bisherigen Wert während des Aufenthaltes im zugehörigen Zustand bei, wobei sich Rücksetzung und Aktivierung nicht beeinflussen, d.h. eine rückgesetzte und nicht aktivierte Uhrvariable weist für die komplette Verweildauer den Wert 0 auf.

Schlussendlich sei jedem Zustand $\mathtt{z}_{\mathtt{i}_\mathtt{k}}$ das – aus Gründen der Übersichtlichkeit gleichnamige – Label $\mathtt{z}_{\mathtt{i}_\mathtt{k}}$ zugeordnet und das Ausgangsalphabet

$\mathtt{W}_\mathtt{i}$ eines jeden Teilautomaten $\mathtt{A}_\mathtt{i}$ als (um ein „Leerelement“ ϵ erweiterte) Teilmenge dieser Label $\mathtt{z}_{\mathtt{i}_\mathtt{k}}$ definiert. Somit gilt implizit, dass die $\mathtt{W}_\mathtt{i}$ (der einzelnen Teilautomaten) paarweise disjunkt sind.

$\mathtt{V}_\mathtt{i}$, das Eingabealphabet des Automaten $\mathtt{A}_\mathtt{i}$, wird als Teilmenge der disjunkten Vereinigung aller Ausgangsalphabete der übrigen Teilautomaten von $\mathtt{A}$ definiert:

$$\mathtt{V}_\mathtt{i} \subseteq \bigcup_{\mathtt{j} \neq \mathtt{i}} \mathtt{W}_\mathtt{j} \tag{5.3}$$

Offensichtlich gilt somit $\mathtt{V}_\mathtt{i} \cap \mathtt{W}_\mathtt{i} = \emptyset$.

Mittels dieser Definition wird es dem (Teil-)Automaten ermöglicht, bei Eintreten einer sich auf das Gesamtsystem beziehenden vordefinierten Situation zu reagieren, wie dies für die Realisierung deterministischer Bedingungen (vgl. Fall (3) in Abschnitt 5.1) notwendig ist. Diese, um die leere Menge $\emptyset$ erweiterte Abhängigkeit wird durch die Menge

$\mathtt{C}_\mathtt{i}$ beschrieben, welche als Menge boolescher Funktionen über dem Eingangsalphabet $\mathtt{V}_\mathtt{i}$ gegeben ist. Hieraus wird jedem abgehenden Zustandsübergang eines jeden Zustandes in $\mathtt{A}_\mathtt{i}$ eine Bedingung $\mathtt{c}_{\mathtt{i}_{\mathtt{k}_\mathtt{j}}}$ zugewiesen. Die $\mathtt{c}_{\mathtt{i}_{\mathtt{k}_\mathtt{j}}}$ eines Zustands $\mathtt{z}_{\mathtt{i}_\mathtt{k}}$ sind hierbei paarweise disjunkt:

$$\mathtt{c}_{\mathtt{i}_{\mathtt{k}_\mathtt{f}}} \wedge \mathtt{c}_{\mathtt{i}_{\mathtt{k}_\mathtt{j}}} = \text{false}, \forall \mathtt{f}, \mathtt{j} \in \{1 \ldots \mathtt{m}\};\ \mathtt{f} \neq \mathtt{j} \tag{5.4}$$

$\mathtt{D}_\mathtt{i}$ ist die – um das abstrakte Nullelement $\diamondsuit$ erweiterte – Menge der, über der Menge der Uhren definierten, Dichtefunktionen. Jede Dichtefunktion ist hierbei über genau einer lokalen Uhr $\mathtt{x}_{\mathtt{i}_\mathtt{g}} \in \mathtt{X}_\mathtt{i}$ definiert und ermöglicht es, die Verzögerungszeiten beschreibenden Übergangstypen (1) und (2) zu modellieren. Hierfür wird jedem abgehenden Übergang eines jeden Zustandes ein Element $\mathtt{d}_{\mathtt{i}_{\mathtt{k}_\mathtt{j}}}$ aus $\mathtt{D}_\mathtt{i}$ zugewiesen. $\mathtt{d}_{\mathtt{i}_{\mathtt{k}_\mathtt{j}}}$ kann, wie in Abschnitt 5.1 erläutert, nur für maximal einen der Übergänge eines jeden Zustandes eine Dichtefunktion enthalten und enthält daher für alle anderen Übergänge statt der Dichtefunktion das Nullelement ($\diamondsuit$):

$$\mathtt{d}_{\mathtt{i}_{\mathtt{k}_\mathtt{j}}} \neq \diamondsuit \Longrightarrow \mathtt{d}_{\mathtt{i}_{\mathtt{k}_\mathtt{f}}} = \diamondsuit, \forall \mathtt{f} \in \{1 \ldots \mathtt{m}\};\ \mathtt{f} \neq \mathtt{j} \tag{5.5}$$

Um die nach Abschnitt 5.1 zulässige ODER-Verknüpfung von mehreren deterministischen Abhängigkeiten mit (maximal) einer stochastischen Verzögerung zu ermöglichen, ist es notwendig, zu zeigen, dass gilt:

Satz 1: Für jeden Zustand $\mathtt{z}_{\mathtt{i}_\mathtt{k}} \in \mathtt{Z}_\mathtt{i}$ lässt sich aus der Disjunktion der Bedingungen $\mathtt{c}_{\mathtt{i}_{\mathtt{k}_\mathtt{j}}}$, $\mathtt{j} = \{1 \ldots \mathtt{m}\}$ und der Dichtefunktion $\mathtt{d}_{\mathtt{i}_{\mathtt{k}_\mathtt{j}}}$ eine gemeinsame, das Verlassen des Zustandes eindeutig bestimmende Dichtefunktion $\mathtt{d}_{\mathtt{i}_\mathtt{k}}(\mathtt{x})$ bilden. $\mathtt{x}$ bezeichne hierbei eine beliebige im Zustand $\mathtt{z}_{\mathtt{i}_\mathtt{k}}$ aktive Uhr.

Beweis: Für jede Dichtefunktion muss gelten:

$$\int_{0}^{\infty} \mathtt{d}_{\mathtt{i}_k}(\mathtt{x})\ d\mathtt{x} = 1 \tag{5.6}$$

Damit ist Satz 1 nur dann gültig, wenn dies auch für die Summe der Zeitintegrale

$$\sum_{\mathtt{j}=1}^{\mathtt{m}} \left(\int_{0}^{\infty} \left\{ \mathtt{c}_{\mathtt{i}_{k_j}} \vee \mathtt{d}_{\mathtt{i}_{k_j}}(\mathtt{x}) \right\} d\mathtt{x} \right) = 1 \tag{5.7}$$

bzw. für das Zeitintegral über der Summe aller abgehenden Übergangsbedingungen

$$\int_{0}^{\infty} \left(\sum_{\mathtt{j}=1}^{\mathtt{m}} \left\{ \mathtt{c}_{\mathtt{i}_{k_j}} \vee \mathtt{d}_{\mathtt{i}_{k_j}}(\mathtt{x}) \right\} \right) d\mathtt{x} = 1 \tag{5.8}$$

gilt. Dies bedeutet, dass obiger Satz genau dann gilt, wenn sich der Ausdruck

$$\sum_{\mathtt{j}=1}^{\mathtt{m}} \left\{ \mathtt{c}_{\mathtt{i}_{k_j}} \vee \mathtt{d}_{\mathtt{i}_{k_j}}(\mathtt{x}) \right\} \text{ in eine Wahrscheinlichkeitsdichtefunktion } \sum_{\mathtt{j}=1}^{\mathtt{m}} \mathtt{d}^{*}_{\mathtt{i}_{k_j}}(\mathtt{x}) \tag{5.9}$$

überführen lässt. Aufgrund dessen, dass $\mathtt{c}_{\mathtt{i}_{k_j}}$ und $\mathtt{d}_{\mathtt{i}_{k_j}}$ definitionsgemäß stochastisch unabhängige Ereignisse darstellen, ist dies jedoch leicht einsichtig. Um selbiges zu zeigen, sei daran erinnert, dass ein Übergang, dessen Bedingung $\mathtt{c}_{\mathtt{i}_{k_j}}$ erfüllt wird, als unverzüglich ausgeführt definiert wurde. Zeitlich ereigne sich die Erfüllung der Bedingungen $\mathtt{c}_{\mathtt{i}_{k_1}}$, $\mathtt{c}_{\mathtt{i}_{k_2}}$, ..., $\mathtt{c}_{\mathtt{i}_{k_m}}$ zu den Zeitpunkten $\mathtt{t}_{\mathtt{e},\mathtt{i}_{k_1}}$, $\mathtt{t}_{\mathtt{e},\mathtt{i}_{k_2}}$, ..., $\mathtt{t}_{\mathtt{e},\mathtt{i}_{k_m}}$, mit $\mathtt{t}_{\mathtt{e},\mathtt{i}_{k_j}} \in [0, \infty)$, was bedeutet, dass jede der Bedingungen aktiviert werden kann. Das Minimum über $\mathtt{t}_{\mathtt{e},\mathtt{i}_{k_1}}$, $\mathtt{t}_{\mathtt{e},\mathtt{i}_{k_2}}$, ..., $\mathtt{t}_{\mathtt{e},\mathtt{i}_{k_m}}$ sei hierbei mit $\mathtt{t}_{\mathtt{e},\mathtt{i}_k}$ bezeichnet, womit sich die, obige Bedingung erfüllende, Definition der zusammengesetzten Dichtefunktion $\mathtt{d}^{*}_{\mathtt{i}_{k_j}}(\mathtt{x})$ ergibt (qed.):

$$\mathtt{d}^{*}_{\mathtt{i}_{k_j}}(\mathtt{x}) = \begin{cases} \mathtt{d}_{\mathtt{i}_{k_j}}(\mathtt{x}) & \text{, für } \mathtt{x} < t_{e,i_k} \\ 0 & \text{, für } \mathtt{x} \geq t_{e,i_k} \text{ und } t_{e,i_{k_j}} \neq t_{e,i_k} \\ \left\{ 1 - \sum_{o=1}^{m} \left(\int_{0}^{t_{e,i_k}} \mathtt{d}_{\mathtt{i}_{k_o}}(\xi)\ d\xi \right) \right\} \cdot \delta(t_{e,i_k}) & \text{, für } \mathtt{x} \geq t_{e,i_k} \text{ und } t_{e,i_{k_j}} = t_{e,i_k} \end{cases} \tag{5.10}$$

Ganz offensichtlich gilt, dass – so lange keine der Bedingungen $\mathtt{c}_{\mathtt{i}_{k_1}}, \mathtt{c}_{\mathtt{i}_{k_2}}, \ldots, \mathtt{c}_{\mathtt{i}_{k_m}}$ erfüllt ist, d.h. $\mathtt{x} < \mathtt{t}_{\mathtt{e},\mathtt{i}_k}$, mit $\mathtt{t}_{\mathtt{e},\mathtt{i}_k} = \mathtt{min}(\mathtt{t}_{\mathtt{e},\mathtt{i}_{k_1}}, \mathtt{t}_{\mathtt{e},\mathtt{i}_{k_2}}, \ldots, \mathtt{t}_{\mathtt{e},\mathtt{i}_{k_m}})$ – die Auslösewahrscheinlichkeit des $\mathtt{j}$-ten Übergangs des aktiven Zustands $(\mathtt{z}_k)$ zu

$$\mathtt{p}_{\mathtt{i}_{k_j}}(\mathtt{x}) = \int_{0}^{\mathtt{x}} \mathtt{d}_{\mathtt{i}_{k_j}}(\xi)\ d\xi \tag{5.11}$$

bestimmt werden kann. □

Die Menge aller im Automat $\mathtt{A_i}$ existenten Zustandsübergänge ist folglich durch $\mathtt{Q_i} = \mathtt{Z_i} \times \mathtt{D_i} \times \mathtt{C_i} \times \mathtt{Z_i}$ eindeutig beschrieben. Hierbei wird erneut deutlich, dass es durchaus möglich ist, über verschiedene Wege vom betrachteten zu einem beliebigen Zustand des i-ten Teilautomaten zu gelangen. Allerdings ist – wie obiger Beweis zeigt – sichergestellt, dass zu keinem Zeitpunkt mehr als ein abgehender Zustandsübergang ermöglicht ist. Hiermit lässt sich dann mit

$\mathtt{Y_i} : \mathtt{Z_i} \times \mathtt{D_i} \times \mathtt{C_i} \times \mathtt{Z_i} \rightarrow \mathtt{W_i}$ die Ausgabefunktion angeben. Diese ordnet jedem Zustandsübergang ein Element des Ausgangsalphabets $\mathtt{W_i}$ zu.

In selbiger Weise lässt sich mit

$\mathtt{L_i} : \mathtt{Z_i} \times \mathtt{D_i} \times \mathtt{C_i} \rightarrow ([0,1] \times \mathtt{Z_i})^{|\mathtt{Z_i}|}$ die Übergangsfunktion angeben. Diese ordnet jedem Zustands-Bedingungs-Paar $(\mathtt{z}_{\mathtt{i_k}}, \mathtt{d}^*_{\mathtt{i_k}}) \in \mathtt{Z_i} \times (\mathtt{D_i} \times \mathtt{C_i})$ die in Abbildung 26 mit $p_{i_{k_{j_h}}} \in [0,1]$ (vgl. Gleichung 5.12) bezeichneten Wahrscheinlichkeiten ebenso zu, wie den jeweils zugehörenden Folgezustand $\mathtt{z}_{\mathtt{i}_{\mathtt{k}_{\mathtt{j_h}}}} \in \mathtt{Z_i}$. Diese Zuordnung entspricht dem direkten Anhängen einer stochastischen Verzweigung (Fall (4), Abschnitt 5.1) und ermöglicht somit die Modellierung des in Abbildung 21 gezeigten allgemeinen Zustandsübergangs.

Während $\mathtt{p}_{\mathtt{i}_{\mathtt{k}_{\mathtt{j}}}}(\mathtt{x})$ (Gleichung 5.11) die Auslösewahrscheinlichkeit eines Zustand-Bedingungspaars $(\mathtt{z}_{\mathtt{i_k}}, \mathtt{d}^*_{\mathtt{i_k}})$ über der Zeit angibt, bezeichnet $p_{i_{k_{j_h}}}$ die (zeitlose) Wahrscheinlichkeit dafür, dass ein auf $\mathtt{Q_i}$ gegebener Übergang stattfindet.

Definition: Sei $\mathtt{z}^\star_{\mathtt{i_k}} \in \mathtt{Z_i}$ ein bei erfülltem $\mathtt{d}^*_{\mathtt{i}_{\mathtt{k}_{\mathtt{j}}}}$ auf $\mathtt{z}_{\mathtt{i_k}}$ folgender Zustand und *next* ein Operator, der den in einem beliebigen Pfad auf $\mathtt{z}_{\mathtt{i_k}}$ folgenden Zustand zurückgibt; dann beträgt die Wahrscheinlichkeit, dass es sich bei diesem Zustand um $\mathtt{z}^\star_{\mathtt{i_k}}$ handelt:

$$p_{i_{k_{j_\star}}} = \mathtt{p}(\mathtt{z}_{\mathtt{i_k}},\ \mathtt{d}^*_{\mathtt{i}_{\mathtt{k}_{\mathtt{j}}}},\ \mathtt{z}^\star_{\mathtt{i_k}}) = \mathrm{Prob}\left(next\,(\mathtt{z}_{\mathtt{i_k}}) = \mathtt{z}^\star_{\mathtt{i_k}} \;\middle|\; \mathtt{d}^*_{\mathtt{i}_{\mathtt{k}_{\mathtt{j}}}} \Rightarrow \mathrm{true}\right) \tag{5.12}$$

Dies impliziert, dass gilt: $$\sum_h p_{i_{k_{j_h}}} = 1 \tag{5.13}$$

Die verbliebene Größe

$\mathtt{X}^{\mathtt{R}}_{\mathtt{i}}$ gibt schließlich an, welche Uhren beim Eintritt in den Folgezustand eines bestimmten Übergangs zurückgesetzt werden: $\mathtt{X}^{\mathtt{R}}_{\mathtt{i}}(\mathtt{x}_{\mathtt{i}_\mathtt{g}},\ \mathtt{q}_{\mathtt{i}_{\mathtt{k}_{\mathtt{j_h}}}}) \in \{-, \mathrm{res}\},\ \mathtt{q}_{\mathtt{i}_{\mathtt{k}_{\mathtt{j_h}}}} \in \mathtt{Q_i},\ \mathtt{x}_{\mathtt{i}_\mathtt{g}} \in \mathtt{X_i}$.

Die auf den kontinuierlichen Automaten angewandte **Semantik** ist wie folgt gegeben. Für zwei Zustände $\mathtt{z1},\ \mathtt{z2} \in \mathtt{Z}_i$ gilt: Wenn der Automat $\mathtt{A_i}$ sich zum Zeitpunkt t_1 im Zustand $\mathtt{z1}$ befindet und es in $\mathtt{L_i}$ einen von Null verschiedenen, zum Zustand $\mathtt{z2}$ führenden Eintrag gibt, dessen über $\mathtt{C_i}$ definierte deterministische Bedingung $\mathtt{c}_{\mathtt{i}_{\mathtt{z1}_\mathtt{j}}}$ zum Zeitpunkt t_1 erfüllt ist (d.h., ein in $\mathtt{c}_{\mathtt{i}_{\mathtt{z1}_\mathtt{j}}}$ referenziertes Ereignis eingetreten ist), tritt dieser Zustandswechsel mit sofortiger Wirkung ein. Der Automat wechselt dann umgehend, mit der durch $\mathtt{L_i}$ gegebenen Wahrscheinlichkeit, in den Zustand $\mathtt{z2}$, wobei die Uhren des Automaten $\mathtt{A_i}$ beim Eintritt in $\mathtt{z2}$ entsprechend $\mathtt{X}^{\mathtt{R}}_{\mathtt{i}}(\mathtt{x}_{\mathtt{i}_\mathtt{g}},\ \mathtt{q}_{\mathtt{i}_{\mathtt{z2}_{\mathtt{j_h}}}})$ modifiziert werden.

Gibt es in $\mathtt{L_i}$ keine solche Intabulation, jedoch eine, deren über $\mathtt{D_i}$ definierte Dichtefunktion $\mathtt{d}_{\mathtt{i}_{\mathtt{z1}_\mathtt{j}}}$ zum Zeitpunkt t_1 einen von Null verschiedenen Wert aufweist, so ist die Wahrscheinlichkeit dafür, dass der Automat in einen anderen Zustand wechselt, durch das Integral dieser Dichtefunktion über der zugehörigen Uhr (von Null bis zum aktuellen Wert) gegeben (vgl. Gleichung 5.11). Dies bedeutet, dass sich die Wahrscheinlichkeit für einen zum Zeitpunkt t_1 stattfindenden Wechsel vom Zustand `z1` in den Zustand `z2` als Produkt aus dem zuvor bestimmten Integralwert und der durch $\mathtt{L_i}$ gegebenen Wahrscheinlichkeit ergibt. Findet der Zustandswechsel statt, werden auch in diesem Fall die Uhren des Automaten $\mathtt{A_i}$ beim Eintritt in `z2` entsprechend $\mathtt{X}_\mathtt{i}^\mathtt{R}(\mathtt{x}_{\mathtt{i}_\mathtt{g}},\ \mathtt{q}_{\mathtt{i}_{\mathtt{z1}_{\mathtt{j_h}}}})$ modifiziert. Findet zum Zeitpunkt t_1 kein Zustandswechsel statt, so verbleibt der Automat im Zustand `z1`, wobei sich die Uhrwerte entsprechend $\mathtt{X}_\mathtt{i}^\mathtt{act}(\mathtt{x}_{\mathtt{i}_\mathtt{g}},\ \mathtt{z1})$ verhalten. ■

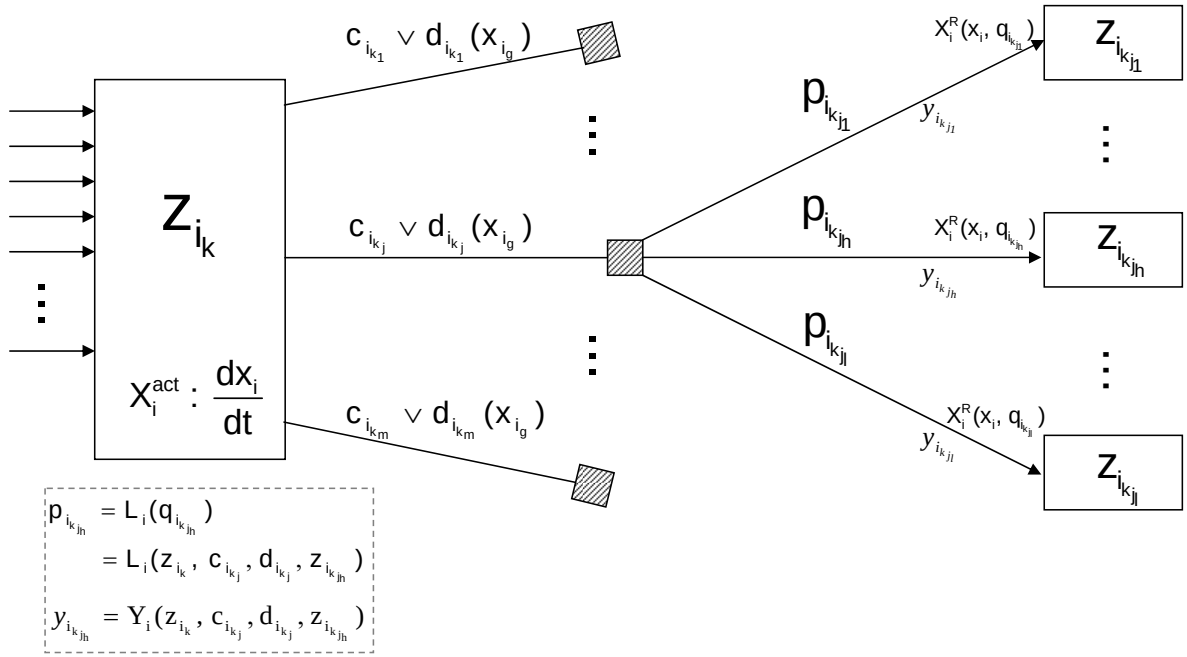

Abbildung 26: Zustand $\mathtt{z}_{\mathtt{i}_\mathtt{k}} \in \mathtt{Z_i}$ samt den in Abschnitt 5.1 eingeführten allgemeinen Zustandsübergängen. Die schraffierten Kästchen zeigen, dass hier ein Zustandsübergang des Typs (4) angehängt wurde. Findet keine Aufsplittung statt ($\mathtt{L}(\mathtt{q}_{\mathtt{i}_{\mathtt{k}_{\mathtt{j_h}}}}) = \mathtt{1}$), kann selbiges in der grafischen Darstellung entfallen. Gleiches gilt für leere Rücksetzbefehle {-}. Wird mit der Auslösung des Übergangs ein Ausgangselement ($y_{\mathtt{i}_{\mathtt{k}_{\mathtt{j_h}}}} \in \mathtt{W_i}$) erzeugt, wird dieses unterhalb des Pfeils notiert. Andernfalls bleibt dieser Platz frei.

Als Beispiel sei der in Abbildung 27 dargestellte Prozess gegeben. Im Grundverhalten liest der Prozess im Zustand `LA` zunächst einen Auftrag ein und arbeitet ihn umgehend ab. Nach 8,5 Sekunden ist dieser Vorgang abgeschlossen und der Prozess gibt das Ergebnis (Zustand `OK`) aus, wofür 1,5 Sekunden benötigt werden, bevor der Prozess wieder in den Zustand `LA` wechselt und den nächsten Auftrag bearbeitet. Während der Bearbeitung ist es möglich, dass das Betriebssystem (Zustand `BS`) für eine stochastisch gegebene Zeitdauer alle Ressourcen anfordert (Ereignis IRQ), weshalb der Prozess so lange unterbrochen, anschließend jedoch an gleichen Stelle fortgesetzt wird. Leider kommt es nur in 99% der Fälle zu einer

korrekten Ausgabe (Zustand OK), wohingegen in 1% der Fälle ein durch das externe Ereignis Rep(ariert) behebbarer Fehler auftritt (Zustand FE). Schlussendlich ist es möglich, dass ein anderer Prozess (Ereignis Abbruch) die Ausgabe (Zustand OK) frühzeitig abbricht. Der Automat geht dann direkt in Zustand LA über. Beim Auslösen des vom Zustand OK (bzw. Zustand FE) nach Zustand LA führenden Übergangs soll das Ausgangselement *OK* (bzw. *FE*) erzeugt werden.

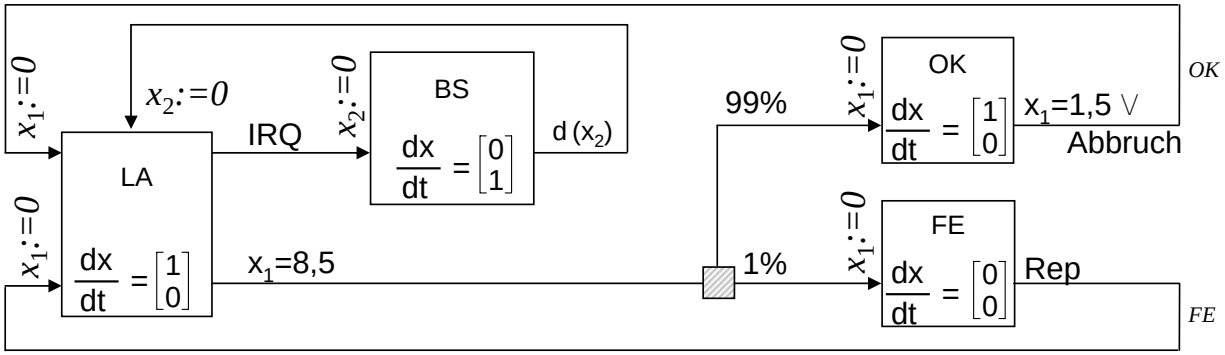

Abbildung 27: Automatendarstellung eines Bearbeitungsprozesses

Hieraus lässt sich die – in Tabelle 4 gezeigte – **formale Automatenrepräsentation** direkt ermitteln. Zunächst werden $\mathtt{Z_i}$, $\mathtt{Pr_i^0}$, $\mathtt{X_i}$, $\mathtt{X_i^R}$ sowie $\mathtt{X_i^{act}}$ in Tabellenform dargestellt. Die erste Spalte zeigt die Menge der Zustände $\mathtt{Z_i}$. In der zweiten Spalte sind hierzu korreliert die Elemente von $\mathtt{Pr_i^0}$ aufgetragen. Die darauf folgenden Spalten sind mit der Menge der Uhren $\mathtt{X_i}$ überschrieben und geben für jede Uhr die zu den einzelnen Zuständen assoziierten Elemente von $\mathtt{X_i^0}$ sowie $\mathtt{X_i^{act}}$ an. $\mathtt{X_i^0}$ gibt an, welchen Wert welche Uhr in welchem Zustand zum Zeitpunkt Null mit welcher Wahrscheinlichkeit einnimmt, wobei $\mathtt{x_1} = \delta(a)$ für einen Dirac-Impuls an $\mathtt{x_1} = a$ und $\mathtt{x_1} = \mathtt{GV}(a, b)$ für eine Gleichverteilung zwischen den Grenzen $(a, b]$ steht. In $\mathtt{X_i^{act}}(\mathtt{x_1})$ ist hinterlegt, dass die Uhr $\mathtt{x_1}$ lediglich in den Zuständen LA und OK aktiv ist, während die Uhr $\mathtt{x_2}$ nur im Zustand BS läuft. Im Zustand FE liegt keine lokale Uhraktivität vor.

Die Ein- und Ausgabealphabete $\mathtt{V_i}$ und $\mathtt{W_i}$ werden zwischen die beiden Teiltabellen in Mengennotation angegeben und leiten zur, die Dynamik des Automaten beschreibenden, zweiten Teiltabelle über. Diese enthält nur jene Elemente aus $\mathtt{L_i}$, für welche die durch $\mathtt{L_i}$ zugewiesene Wahrscheinlichkeit größer Null ist. In der ersten Spalte findet sich dabei der aktuelle Zustand und in der zweiten die zeitbasierte sowie in der dritten Spalte die ereignisbasierte Bedingung. In den folgenden Spalten werden dann die zugeordneten Größen in folgender Reihenfolge eingetragen: nachfolgender Zustand ($\mathtt{Z}(\mathtt{L_i})$), Wahrscheinlichkeit ($\mathtt{p}(\mathtt{L_i})$), erzeugtes Ausgangssymbol ($\mathtt{Y_i}$) und die Rücksetzanweisungen ($\mathtt{X_i^R}$), wobei letztere als transponierter, den Uhren aus ($\mathtt{X_i}$) elementweise zugeordnetem, Vektor geschrieben werden. Beispielhaft wird die zweite Zeile (von oben) wie folgt gelesen: Befindet sich der Automat im Zustand LA und hat die Uhr $\mathtt{x_1}$ den Wert 8,5 angenommen, so wechselt der Automat mit einer Wahrscheinlichkeit von 99% in den Zustand OK, wobei die Uhr $\mathtt{x_1}$ auf Null zurückgesetzt wird und die Uhr $\mathtt{x_2}$ ihren aktuellen Wert beibehält.

Tabelle 4: Formale Darstellung des kontinuierlichen Automaten aus Abbildung 27

$\mathtt{Z_i}$	$\mathtt{Pr_i^0}$	$\mathtt{X_i} = \{\mathtt{x_1}, \mathtt{x_2}\}$			
		$\mathtt{X_i^0}(\cdot, \mathtt{x_1})$	$\mathtt{X_i^{act}}(\cdot, \mathtt{x_1})$	$\mathtt{X_i^0}(\cdot, \mathtt{x_2})$	$\mathtt{X_i^{act}}(\cdot, \mathtt{x_2})$
LA	85%	GV(0, 8,5)	1	$\delta(0)$	0
BS	0	$\delta(0)$	0	$\delta(0)$	1
OK	15%	GV(0, 1,5)	1	$\delta(0)$	0
FE	0	$\delta(0)$	0	$\delta(0)$	0

$$\mathtt{V}_i = \{\text{ IRQ, Rep, Abbruch}\}$$
$$\mathtt{W}_i = \{-, \mathit{FE}, \mathit{OK}\}$$

$\mathtt{Z_i}$	$\mathtt{D_i}$	$\mathtt{C_i}$	$\mathtt{Z}(\mathtt{L_i})$	$p(\mathtt{L_i})$	$\mathtt{Y_i}$	$\mathtt{X_i^R}$
LA	$x_1 = 8{,}5$	–	FE	0,01	–	$(res, -)^T$
			OK	0,99	–	$(res, -)^T$
LA	–	IRQ	BS	1	–	$(-, res)^T$
BS	$\mathrm{d}(x_2)$	–	LA	1	–	$(-, res)^T$
FE	–	Rep	LA	1	FE	$(res, -)^T$
OK	$x_1 = 1{,}5$	Abbruch	LA	1	OK	$(res, -)^T$

5.4 Zeitdiskreter Automat

Aufgrund der tendenziell beschränkten Rechenleistung, stößt eine direkte Implementierung von PTAs sehr schnell an ihre Grenzen, weshalb der Zustandsraum einer deutlichen Reduktion bedarf. Im Rahmen dieser Arbeit wird dies durch Beschränkung der Zeitachse auf eine Menge diskreter Zeitpunkte erreicht, da die auf DTMCs basierende Form des PMCs – wie in Kapitel 3 erläutert – nur zeitdiskrete Modelle unterstützt. Im Gegensatz zu der von [Henzinger et al., 1992] vorgeschlagenen Diskretisierung mittels digitaler Uhren (digital clocks), welche die Diskretisierung erst vor dem Analyseschritt durch PMC über das komplette Modell durchführen würde, entsteht durch direkte Überführung der Teilmodul-PTAs ein diskretes Modell, welches jedem Zustand eine diskrete und in Richtung der Zu-

standsübergänge monoton steigende Zeitinformation zuordnet. Die Transformation in eine formale Sprache kann dann durch einfaches Hinzufügen eines Synchronisationsimpulses erreicht werden. Da alle Teilmodule stets aktuelle Werte aufweisen sollen, ist es erforderlich, dass jedes Modul beim Eintreffen des Synchronisationsimpulses seinen aktuellen Zustand ermittelt (auch wenn sich dieser nicht ändert).

Der zu verwendende zeitdiskrete Automat beschreibt daher ein durch einen externen Uhrimpuls gesteuertes System, wobei der Abstand zweier Uhrimpulse die Länge Δt hat. Die formale **Definition** des diskreten Automaten ist durch

$$\mathbf{A} = \{\mathbf{A_i} \mid i = 1 \ldots n\}, \quad \text{mit} \tag{5.14}$$

$$\mathbf{A_i} = \left(\mathbf{Z_{p_i}}, \mathbf{z^0_{p_i}}, \mathbf{X_i}, \mathbf{x^0_i}, \mathbf{Z_{t_i}}, \mathbf{Q_i}, \mathbf{W_i}, \mathbf{V_i}, \mathbf{c_i}, \mathbf{p_i}, \mathbf{f_i}\right) \tag{5.15}$$

gegeben. Der Automat $\mathbf{A}$ weist eine Rückführautomatenstruktur auf, was bedeutet, dass jeder Teilautomat $\mathbf{A_i}$ den tatsächlichen Zustand des Gesamtautomaten $\mathbf{A}$ kennen kann (falls dies gewünscht ist). Die einzelnen Elemente eines jeden Teilautomaten (vgl. Abbildung 28) sind wie im Folgenden beschrieben definiert, wobei – soweit als möglich – die Bezeichnungen entsprechend jenen des kontinuierlichen Automaten gewählt wurden:

$\mathbf{Z_{p_i}}$ Menge der persistenten Zustände. Unter einem persistenten Zustand versteht man einen (in diesem Fall diskreten) Zustand, in dem die Verweilzeit mindestens ein Δt groß ist. Im Gegensatz hierzu stehen die transienten (i.S.v. vorübergehenden) Zustände, die keinerlei Aufenthaltsdauer besitzen. $|\mathbf{Z_{p_i}}|$ wird als endlich angenommen. Die Elemente von $\mathbf{Z_{p_i}}$ werden mit $\mathbf{z_{i_k}}$ bezeichnet: $\mathbf{Z_{p_i}} = \{\, \mathbf{z_{i_k}} \mid k = 1, \ldots, n\}$

$\mathbf{z^0_{p_i}}$ ordnet jedem persistenten Zustand $\mathbf{z_{i_k}} \in \mathbf{Z_{p_i}}$ eine Wahrscheinlichkeit dafür zu, dass es sich bei diesem Zustand um den Anfangszustand, d.h. um den zum Zeitpunkt $t = 0$ aktiven Zustand, handelt: $\mathbf{z^0_{p_i}} : \mathbf{Z_{p_i}} \rightarrow [\mathbf{0}, \mathbf{1}]$ mit $\sum_{\mathbf{z_{i_k}} \in \mathbf{Z_{p_i}}} \mathbf{z^0_{p_i}}(\mathbf{z_{i_k}}) = \mathbf{1}$.

$\mathbf{X_i}$ Menge lokaler Uhren. Der Wertebereich der Uhren besteht aus positiven ganzzahligen Vielfachen von Δt. Die, diese Vielfachen von Δt repräsentierenden, Elemente von $\mathbf{X_i}$ werden mit $\mathbf{x_{i_g}} \in \mathbb{N}_0$ bezeichnet.

$\mathbf{x^0_i}$ Abbildung, die jeder lokalen Uhr für jeden Zustand $\mathbf{z_{i_k}} \in \mathbf{Z_{p_i}}$ einen Anfangswert zuweist. Für jedes $\mathbf{x_{i_g}} \in \mathbf{X_i}$ ist der stochastisch gesetzte Anfangswert $\mathbf{x^0_{i_g}}(\mathbf{z_{i_k}}) \in \mathbb{N}_0$ mittels der durch $\mathbf{x^0_i}$ gegebenen Häufigkeitsfunktion $\mathbb{N}_0 \rightarrow [0, 1]$ festgelegt.

$\mathbf{Z_{t_i}}$ Menge der transienten Zustände, wobei $\mathbf{Z_{p_i}} \cap \mathbf{Z_{t_i}} = \emptyset$ gilt.
$|\mathbf{Z_{t_i}}|$ wird als abzählbar unendlich angenommen, da nur so eine Transformation vom kontinuierlichen zum diskreten Automaten mit beliebig (kleinem) Δt garantiert werden kann. Die Notwendigkeit, $|\mathbf{Z_{t_i}}|$ im Rahmen der späteren Implementierung auf einen endlichen Wert zu beschränken, bleibt hiervon unberührt.

$\mathbf{Q_i}$ Menge der Zustandsübergänge $\mathbf{Q_i} \subseteq ((\mathbf{Z_{p_i}} \times \mathbf{Z_{t_i}}) \cup (\mathbf{Z_{t_i}} \times \mathbf{Z_{p_i}}))$. Sie beinhaltet sämtliche möglichen, zwei Zustände miteinander verbindenden Übergänge, wobei immer nur Übergänge von einer in die andere der beiden Zustandsmengen (und umgekehrt) zulässig sind („bipartite graph“). $|\mathbf{Q_i}|$ ist damit ebenfalls abzählbar unendlich.

$\mathbf{W_i}$ Menge boolescher Ausgangsvariablen, welche über einer Teilmenge der Zustandsübergänge $\mathbf{Q_i}$ definiert ist. Die Ausgangsvariablenmengen zweier Teilautomaten sind somit disjunkt zueinander.

$\mathbf{V_i}$ Menge boolescher Eingangsvariablen mit $\mathbf{V_i} \subseteq \bigcup_{\mathbf{j \neq i}} \mathbf{W_j}$

$\mathbf{c_i}$: $\mathbf{Q_i} \cap (\mathbf{Z_{p_i}} \times \mathbf{Z_{t_i}}) \rightarrow \mathbf{C}$ ist die Übergangsbedingung, wobei $\mathbf{C}$ die Menge boolescher Funktionen der Form $\mathbf{c_i} = \mathbf{c_{i_V}} \wedge \mathbf{c_{i_X}}$ darstellt. $\mathbf{c_{i_V}}$ ist hierbei eine boolesche Funktion über der Menge der Eingangsvariablen $\mathbf{V_i}$ und $\mathbf{c_{i_X}}$ ist eine boolesche Funktion über der Menge der Uhrwächter (engl.: time guards). Unter einem Uhrwächter versteht man dabei eine Relation der Form $\mathbf{x_i} \lozenge \mathbf{a}$ mit $\mathbf{x_i} \in \mathbf{X_i}$, $\mathbf{a} \in \mathbb{N_0}$ und $\lozenge \in \{=, \leq, <\}$. Für die, an die von einem persistenten Zustand $\mathbf{z_{i_k}} \in \mathbf{Z_{p_i}}$ abgehenden Übergänge geschriebenen, Bedingungen $\mathbf{c_{i_{k_j}}}$ muss gelten, dass

1. keine zwei Übergangsbedingungen zur selben Zeit erfüllt sind:

$$\mathbf{c_{i_{k_l}}} \wedge \mathbf{c_{i_{k_j}}} = \text{false}, \forall l, j \in \{1 \ldots m\};\ l \neq j \tag{5.16}$$

2. stets mindestens eine Bedingung erfüllt ist:

$$\bigvee_{\mathbf{j}} \mathbf{c_{i_{k_j}}} = \text{ true} \tag{5.17}$$

Falls Gleichung 5.17 nicht erfüllt ist, ist der Automat nicht vollständig, weshalb in diesem Fall ein weiterer, Gleichung 5.17 saturierender Übergang eingeführt werden muss, mit

$$\mathbf{c_{i_{k_{m+1}}}} = \neg \left(\bigvee_{j=1}^{m} \mathbf{c_{i_{k_j}}} \right) \tag{5.18}$$

Dies leitet sich aus der Überlegung ab, dass, falls beim Eintreffen des systemweiten Schaltimpulses keiner der abgehenden Übergänge aktiv wäre, der Automat in seinem bisherigen Zustand verbliebe und lediglich die interne Uhr um einen Zeitschritt erhöht würde. Für die Implementierung würde dies jedoch bedeuten, dass zwischen dem Verhalten des Automaten mit und ohne aktivierter Bedingung unterschieden werden müsste. Erweitert man hingegen den Zustand um obigen Zustandsübergang, welcher auf den bisherigen Zustand zurückgeführt und genau dann aktiv ist, wenn keiner der anderen abgehenden Übergänge aktiv ist, kann die Implementierung des Weiterschaltens als synchron zu einem systemweiten Uhrimpuls erfolgen.

$\mathbf{p_i}$: $\mathbf{Q_i} \cap (\mathbf{Z_{t_i}} \times \mathbf{Z_{p_i}}) \rightarrow [\mathbf{0}, \mathbf{1}]$ repräsentiert die Übergangswahrscheinlichkeit.
Für die Wahrscheinlichkeiten, welche an die von einem transienten Zustand $\tilde{\mathbf{z}}_{\mathbf{i_{k_j}}} \in \mathbf{Z_{t_i}}$ abgehenden Übergänge geschrieben werden, muss gelten:

$$\sum_{\mathbf{h}} \mathbf{p_{i_{k_{j_h}}}} = 1 \quad \forall k, j \tag{5.19}$$

$\mathbf{f_i}$ Uhrmodifikationsfunktion. Diese ordnet jedem von einem transienten zu einem persistenten Zustand verlaufenden Übergang einen Vektor zu, welcher für jede im i-ten Teilautomaten definierte Uhr angibt, ob diese bei Eintritt in den persistenten Zustand um eins erhöht (*inc*), zurückgesetzt (*res*) oder nicht verändert (–) wird.
$\mathbf{f_i} : \mathbf{Q_i} \cap (\mathbf{Z_{t_i}} \times \mathbf{Z_{p_i}}) \rightarrow \{\ inc,\ res,\ -\ \}^{|\mathbf{X_i}|}$

Zusammengenommen entsteht die folgende **Semantik**: Gegeben seien zwei persistente Zustände $\mathbf{z1},\ \mathbf{z2} \in \mathbf{Z_{p_i}}$, die über einen transienten Zustand $\mathbf{zt} \in \mathbf{Z_{t_i}}$ verbunden sind (d.h. $\mathbf{z1} \times \mathbf{zt} \in \mathbf{Q_i}$ und $\mathbf{zt} \times \mathbf{z2} \in \mathbf{Q_i}$).
Befindet sich der Automat $\mathbf{A_i}$ zum durch $r \cdot \Delta t < t_1 < (r+1) \cdot \Delta t$ gegebenen Zeitpunkt t_1 im Zustand $\mathbf{z1}$ und ist die Bedingung $\mathbf{c_i}(\mathbf{z1} \times \mathbf{zt})$ erfüllt, dann wechselt der Automat mit Eintreffen des systemweiten Schaltimpulses ($t_2 = (r+1) \cdot \Delta t$) in den transienten Zustand $\mathbf{zt}$ und von dort umgehend in einen der nachfolgenden[3] persistenten Zustände. Zum Zeitpunkt $t_3 = t_1 + \Delta t$ befindet sich der Automat also mit der Wahrscheinlichkeit $\mathbf{p_i}(\mathbf{zt} \times \mathbf{z2})$ im Zustand $\mathbf{z2}$, wobei die Uhren des Automaten entsprechend $\mathbf{f_i}(\mathbf{zt} \times \mathbf{z2})$ geändert worden sind.

Abbildung 28 zeigt die **grafische Darstellung** des diskreten Automaten für einen einzelnen Zustand $\mathbf{z_{i_k}} \in \mathbf{Z_{p_i}}$. Transiente Zustände werden als kleine schraffierte Quadrate dargestellt, persistente Zustände hingegen als (nicht-ausgefüllte) Rechtecke, in welche die Zustandsbezeichnung geschrieben wird. Besitzt ein transienter Zustand nur einen abgehenden Übergang ($\mathbf{p}$=1), so muss der transiente Zustand nicht dargestellt und der Uhrmodifikationsvektor kann an den verbliebenen (ankommenden) Übergang geschrieben werden. Eine immer erfüllte Bedingung („true“) darf in der grafischen Darstellung ebenso weggelassen werden, wie der an den Übergangsspitzen zu notierende Uhrmodifikationsoperator $\{-\}$ einzelner Uhren.

Zur kompakteren Darstellung ist es darüber hinaus möglich, die reinen, d.h. wahrscheinlichkeitsfreien, Selbstschleifen in der grafischen Repräsentation wegzulassen und innerhalb der Zustände zu vermerken, welche Uhren ggf. um eins erhöht werden. ■

Abbildung 29 zeigt die mit $\Delta t = 1$ diskretisierte Version des in Abbildung 27 dargestellten zeitkontinuierlichen Automaten. Ein Vergleich der beiden Abbildungen lässt erkennen, dass strukturelle Änderungen – abgesehen von den bereits motivierten Selbstschleifen – nur dort auftreten, wo im Kontinuierlichen Wahrscheinlichkeitsdichten verwendet worden sind. Bei dem im Beispiel der Abbildung 29 hiervon betroffenen, vom Zustand `BS` abgehenden Übergang, wurde aus Gründen der Übersichtlichkeit eine abstrakte Darstellung gewählt, welche eine von der diskreten Zeit $\mathbf{x}_2$ abhängige Wahrscheinlichkeit $p_{BS}(x_2)$ verwendet. Eine mögliche explizite Umsetzung findet sich in Abbildung 32.

[3] Ein persistenter Zustand $\mathbf{zP}$ gilt als – in Bezug auf einen transienten Zustand $\mathbf{zT}$ – nachfolgend, falls $\mathbf{zT} \times \mathbf{zP} \in \mathbf{Q_i}$

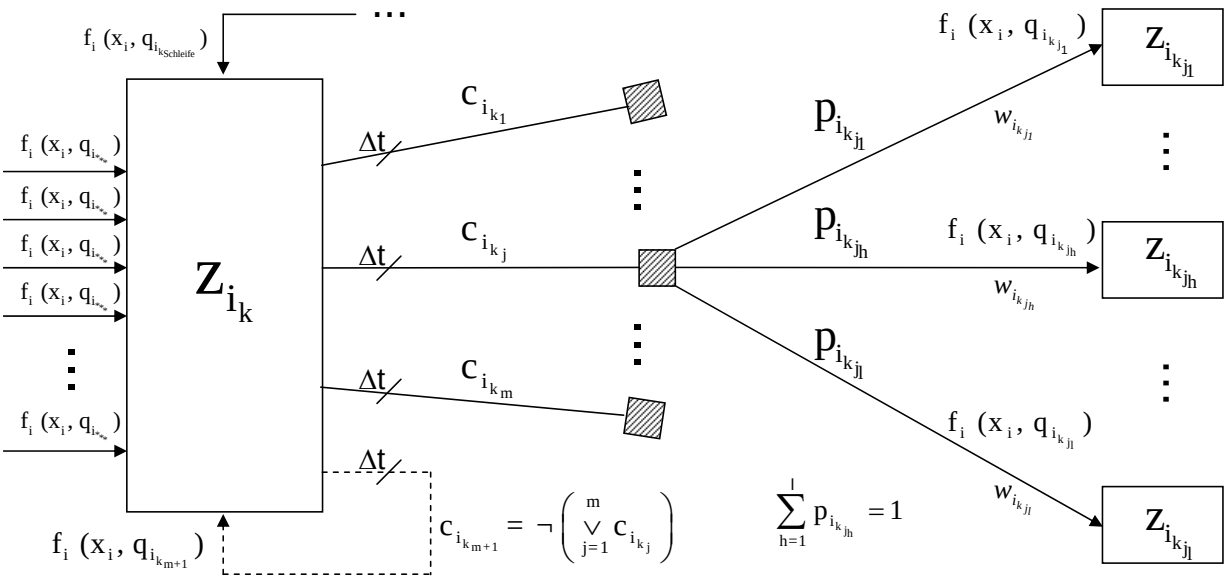

Abbildung 28: Wahrscheinlichkeitsbasierter zeitbewerteter Automat in zeitdiskreter Darstellung

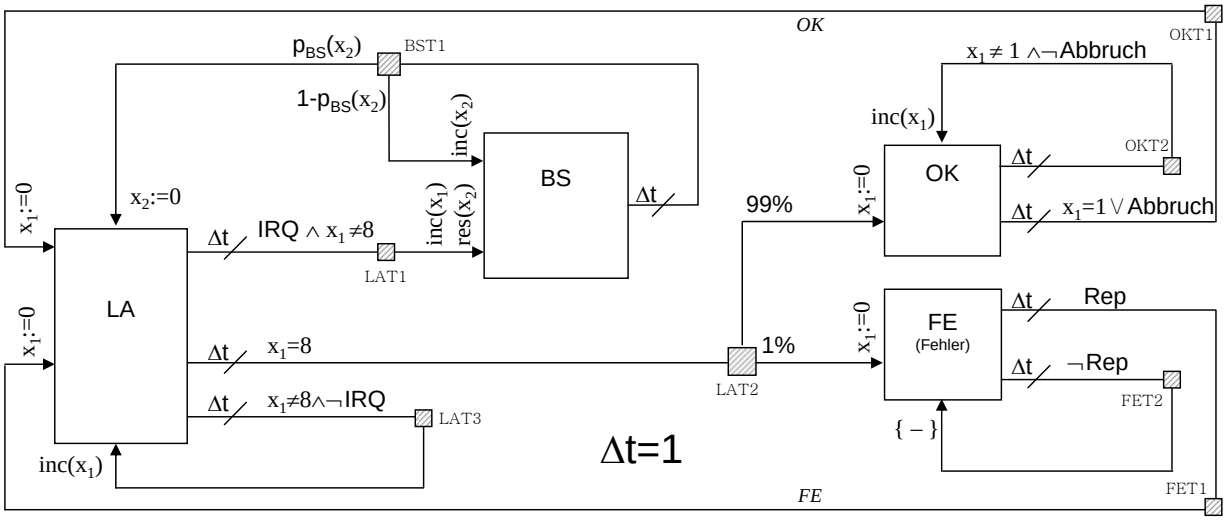

Abbildung 29: Zeitdiskrete Version des Bearbeitungsprozesses aus Abbildung 27. Die Diskretisierungsschrittweite wurde zu $\Delta t = 1$ gewählt.

Die zur **formalen Darstellung** verwendete Notation (Tabelle 5) ist, wie im kontinuierlichen Fall, aus drei Teilen aufgebaut. In der oberen Tabelle finden sich $\mathbf{Z_{p_i}}$, $\mathbf{z^0_{p_i}}$, $\mathbf{X_i}$, sowie $\mathbf{X^0_i}$. Die erste Spalte zeigt die Menge der Zustände $\mathbf{Z_i}$. In der zweiten Spalte sind die hierzu korrelierten Elemente von $\mathbf{z^0_{p_i}}$ aufgetragen. Die darauf folgenden Spalten sind mit der Menge der Uhren $\mathbf{X_i}$ überschrieben und geben für jede Uhr die zu den einzelnen Zuständen assoziierten Elemente von $\mathbf{x^0_i}$ an. Die Notation `[0, 8]: 1/9` besagt, dass alle im Intervall von 0 bis 8 vorkommenden Zahlenwerte mit der Wahrscheinlichkeit $\frac{1}{9}$ auftreten. Sind unterschiedliche Wahrscheinlichkeiten zuzuweisen, sind die einzelnen Instruktionen durch Strichpunkte zu trennen (Bsp: `[0, 3]: 1/5; [4, 5]: 1/10`). Zwischen den beiden Teiltabellen steht wiederum die Menge des Eingabealphabets.

Tabelle 5: Formale Darstellung des diskreten Automaten aus Abbildung 29

$\mathbf{Z_i}$	$\mathbf{z^0_{p_i}}$	$\mathbf{X_i} = \{\mathbf{x_1}, \mathbf{x_2}\}$	
		$\mathbf{x^0_i}(\mathbf{x_1})$	$\mathbf{x^0_i}(\mathbf{x_2})$
LA	$\frac{9}{11}$	$[0,\ 8] : \frac{1}{9}$	$[0] : 1$
BS	0	$[0] : 1$	$[0] : 1$
OK	$\frac{2}{11}$	$[0,\ 1] : \frac{1}{2}$	$[0] : 1$
FE	0	$[0] : 1$	$[0] : 1$

$\mathbf{V_i}$ = { IRQ, Abbruch, Reset }

$\mathbf{Z_{p_i}}$	$\mathbf{c_i}$	$\mathbf{Z_{t_i}}$	$\mathbf{p_i}$	$\mathbf{W_i}$	$\mathbf{f_i}$	$\mathbf{Z_{p_i}}$
LA	IRQ $\wedge$ $\mathbf{x_1} \neq 8$	LAT1	1	–	$(inc, res)^T$	BS
LA	$\mathbf{x_1} = 8$	LAT2	0,99 0,01	– –	$(res, -)^T$ $(res, -)^T$	OK FE
LA	$\mathbf{x_1} \neq 8 \wedge \neg$IRQ	LAT3	1	–	$(inc, -)^T$	LA
BS	*true*	BST1	$p_{BS}(\mathbf{x_2})$ $1 - p_{BS}(\mathbf{x_2})$	– –	$(-, res)^T$ $(-, inc)^T$	LA BS
OK	$\mathbf{x_1} = 1 \ \vee$ Abbruch	OKT1	1	*OK*	$(res, -)^T$	LA
OK	$\mathbf{x_1} \neq 1 \wedge \neg$Abbruch	OKT2	1	–	$(inc, -)^T$	OK
FE	Rep	FET1	1	*FE*	$(res, -)^T$	LA
FE	$\neg$Rep	FET2	1	–	$(-, -)^T$	FE

In der unteren Teiltabelle findet sich die Beschreibung der Automatendynamik und hierin implizit enthalten die Menge der Zustandsübergänge $\mathbf{Q_i}$, wobei auf der linken Seite Übergänge von einem persistenten in einen transienten und auf der rechten Seite die Übergänge von einem transienten in einen persistenten Zustand aufgeführt sind. Die erste Spalte der Teiltabelle enthält persistente Zustände $\mathbf{Z_{p_i}}$ und die zweite Spalte die mit den abgehenden Übergängen assoziierten Übergangsbedingungen $\mathbf{c_i}$. Die dritte Spalte enthält den transienten Zustand $\mathbf{Z_{t_i}}$, der von $\mathbf{Z_{p_i}}$ erreicht wird, falls $\mathbf{c_i}$ erfüllt ist, bzw. von dem aus zum nächsten transienten Zustand weitergeschaltet wird. Da die transienten Zustände

stets direkt mit dem vorherigen persistenten Zustand verkoppelt sind (d.h., es gibt immer nur einen persistenten Zustand, von dem aus man zu einem bestimmten transienten Zustand gelangen kann und es existiert hierfür immer genau ein Übergang), wurde die von den transienten Zuständen ausgehende Wahrscheinlichkeitsaufsplittung als zum Übergang vom persistenten zum transienten Zustand gehörend dargestellt und insofern in derselben Zeile notiert.
In der folgenden vierten Spalte findet sich daher zunächst die Eintrittswahrscheinlichkeit $\mathbf{p_i}$, gefolgt von der zu aktivierenden Ausgangsvariable ($\mathbf{W_i}$, fünfte Spalte), welche angibt, ob und in welcher Form die Auslösung dieses Übergangs für andere Teilautomaten sichtbar sein soll. Die vorletzte Spalte enthält die Angabe der Uhrmodifikationen ($\mathbf{f_i}$). Und die letzte Spalte benennt den nachfolgenden persistenten Zustandes ($\mathbf{Z_{p_i}}$). Für die Uhrmodifikationen wurde hierbei die bereits vom kontinuierlichen Automaten her bekannte mit den Elementen aus $\mathbf{X_i}$ assoziierte transponierte Vektordarstellung verwendet.

5.5 Diskretisierung

5.5.1 Prozessaggregation

Nach der Diskretisierung repräsentiert das Modell nicht mehr den exakten Eintrittszeitpunkt eines Ereignisses, sondern lediglich die Tatsache, dass sich ein Ereignis im vergangenen Zeitintervall ereignet hat. In anderen Worten bedeutet dies, dass sich nach der Diskretisierung zwischen zwei Takten keine Zustandsänderung ergibt, weshalb sich die Übergangswahrscheinlichkeit als Integral über der Zeit mit Breite der Diskretisierungsschrittweite berechnen lässt (vgl. Gleichung 5.20).

In Systemen, die digitale Komponenten enthalten, ist der genaue (i.S.v. reellwertige) Eintrittszeitpunkt meist nur von untergeordnetem Interesse, da diese dem System selbst eine diskrete Zeitachse aufzwingen. In der Mehrzahl der Anwendungen ist es darüber hinaus uninteressant, ob bei zwei sehr schnell aufeinander folgenden Ereignissen das eine oder das andere Ereignis zuerst aufgetreten ist, falls sich diese in voneinander unabhängigen Systembereichen befinden. Beispielsweise spielt es für das Gesamtsystem i.d.R. keine Rolle, welche von zwei SPSen ihre Werte zuerst einliest.

Insbesondere in Bezug auf eine deutliche Zustandsraumreduktion und Berechnungszeitverminderung ist es in diesen Fällen von großem Vorteil, dass durch den vorgeschlagenen diskreten Automaten lediglich die Tatsache, dass sich ein Ereignis im letzten Zeitschritt ereignet hat, aufgezeichnet wird, nicht aber der exakte Eintrittszeitpunkt. Dies eröffnet auch die Möglichkeit, das mit Klärung der tatsächlichen Reihenfolge behaftete oftmals nicht-deterministische oder stochastische Problem von nicht miteinander in direktem Kontakt stehenden Nebenläufigkeitsproblemen bei nahezu gleichzeitigen Ereigniseintritten in ein deterministisches Problem zu überführen. Hierbei wird die für das Gesamtverhalten irrelevante Information der tatsächlichen Reihenfolge vernachlässigt (i.S.v. verborgen) und somit die Größe des entstehenden Gesamtautomaten erheblich reduziert.

Durch die Diskretisierung kann allerdings auch die zeitliche Trennung von Ereignissen verschwinden, die sich in gekoppelten Bereichen wiederfinden. Hierdurch kann es passieren, dass ein diskreter Zustand mehr als einen abgehenden Übergang mit erfüllter Bedingung aufweist. Eine solche Situation muss dadurch aufgelöst werden, dass jeder der betroffenen Übergangsbedingungen die Negation der anderen betroffenen Bedingung mit logischem UND angehängt wird und für alle möglichen Kombinationen eigenständige Zustandsübergänge eingeführt werden. Sind z.B. zwei Übergangsbedingungen `A` und `B` betroffen, werden die Bedingungen dieser beiden Zustandsübergänge in `A&!B` sowie `!A&B` umgewandelt und ein weiterer Übergang eingefügt, der den Fall `A&B` behandelt. Die Analyse letzteren Falles ist jedoch selten trivial, und sollte daher mit äußerster Umsichtigkeit angegangen werden. Wichtig ist es in diesem Zusammenhang zu erwähnen, dass es häufig Fälle gibt, in denen es nicht vorgesehen ist, dass diese beiden Ereignisse direkt nacheinander ausgeführt werden. In einem solchen Fall muss entweder mit stochastischen Eintrittswahrscheinlichkeiten gearbeitet oder eine Priorität festgelegt werden. Im Automaten der Abbildung 27 trifft dies beispielsweise für den Zustand `LA` zu: Aus der kontinuierlichen Darstellung geht nicht hervor, ob $\mathbf{x}_1 = 8$ oder IRQ höher zu werten sind. Um jedoch eine über den Maximalwert von 8 hinausgehende Erhöhung der Uhr $\mathbf{x}_1$ zu verhindern, macht es (unter der Annahme, dass IRQ ein eher seltenes Ereignis ist) durchaus Sinn, $\mathbf{x}_1 = 8$ über IRQ zu stellen. Wollte man hingegen IRQ über $\mathbf{x}_1 = 8$ stellen, müsste der Automat strukturell geändert werden (Abbildung 30).

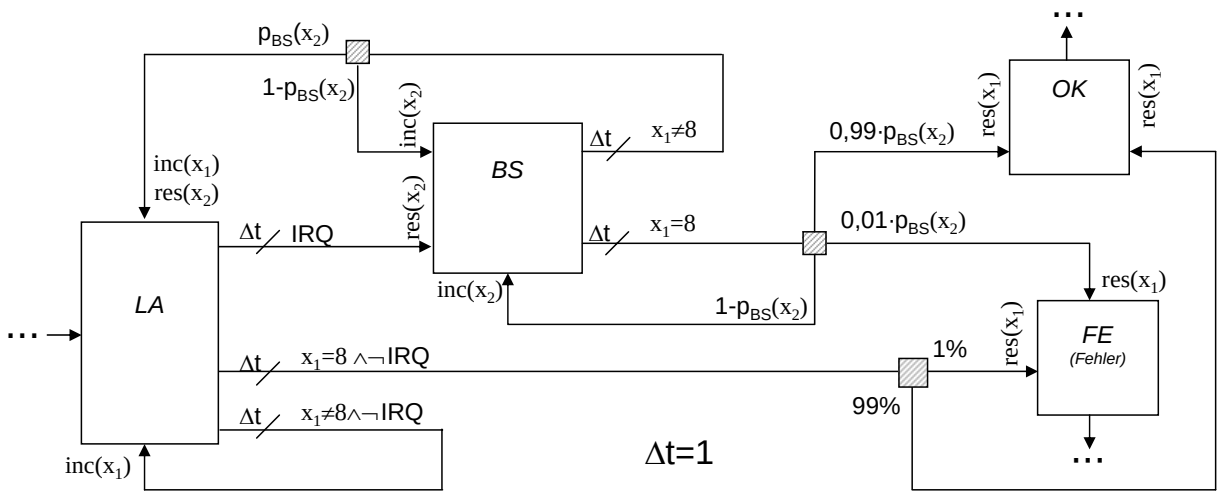

Abbildung 30: Ausschnitt aus dem Automaten, der entstünde, wenn das Ereignis IRQ dominant gegenüber $\mathbf{x}_1 = 8$ zu behandeln wäre.

Anmerkung: Während es im kontinuierlichen Fall keine Rolle spielt, wenn ein Ereignis eine ganze Kette von direkt aufeinander aufbauenden Ereignissen auslöst, ist dies im diskreten Fall getrennt zu behandeln, da dort die Weiterleitung eines jeden Ereignisses einen Zeitschritt in Anspruch nimmt: Eine Kette von Ereignissen, die im Kontinuierlichen in verschwindend geringer Zeit ablief, würde im Diskreten eine ganze Weile dauern. Am einfachsten ist es, wenn sich die Ereigniskette als ein einzelnes Ereignis darstellen lässt und

nur dieses in den diskreten Automaten aufgenommen werden muss. Falls ein – zumindest teilweises – Zusammenfassen nicht möglich ist, muss geprüft werden, ob z.B. mit angehaltenen Uhren gearbeitet werden kann. Eine solche Lösung sollte allerdings nur das letzte Mittel sein, da hierfür Zeitinformationen über alle Module hinweg ausgetauscht werden müssten.

5.5.2 Zeitschrittweite

Für die Charakteristik des diskreten Synchronisationsimpulses werden zwei verschiedene Varianten vorgeschlagen:

1. Ein einheitlicher Systemuhrimpuls, welcher eine konstante Zeitdauer hat und auf jedes Teilmodul als Schaltimpuls wirkt. Dies würde den zeitbewerteten Automaten – abstrakt betrachtet – auf einen um eine Zeitvariable ergänzten klassischen Automaten reduzieren. In diesem Falle ist es nicht zwingend erforderlich (wohl aber in den meisten Fällen sinnvoll), die Zeitinformation explizit zu jedem Zustand zu kodieren, da diese Information durch eine Zählung der Impulse und somit der ausgeführten Zustandsübergänge indirekt ermittelt werden kann.

2. Ein ereignisbasierter Systemuhrimpuls, dessen Länge aufgrund des aktuellen Systemzustandes ermittelt wird und dem (auf volle Zeittakte aufgerundeten) zeitlichen Abstand bis zum nächsten zu erwartenden Ereignis entspricht.

Im Rahmen dieser Arbeit wurde von einer konstanten Schrittweite ausgegangen, wobei im Abschnitt 5.7 eine Erweiterung hin zu einem ereignisbasierten Systemuhrimpuls auf Basis eines Vielfachens einer fixen Schrittweite vorgestellt wird.

In digitalen Schaltungen, bei denen alle Systemänderungen synchron zu einem systemweiten Synchronsignal erfolgen, wird die Länge der minimalen Zeitschrittweite quasi ererbt. Werden hingegen zeitkontinuierliche physikalische Schnittstellen (Sensoren) behandelt, gibt es weder eine obere noch eine untere Schranke, um die Frage der Zeitschrittweite zu beantworten. In manchen Systemen ist es sogar unmöglich, eine als optimal angesehene Zeitschrittweite in der Weise zu bestimmen, dass jedes beteiligte Teilsystem genau eine Aktion ausführen kann. Wählt man diese Schrittweite länger als den Abstand des kürzesten Zustandswechsels, ist Informationsverlust unvermeidbar. Wählt man die Zeitschrittweite hingegen so, dass sie kürzer ist, als der kürzeste zeitliche Abstand zweier Systemänderungen, wird sich dies in der Modellgröße mit exponentieller Verstärkung widerspiegeln.

Darüber hinaus muss berücksichtigt werden, dass die Teilkomponenten eines NAS nichtsynchrone, unabhängige Uhren haben, d.h., Abweichungen in der Uhrgenauigkeit unvermeidlich sind. Will man dies nicht explizit im Modell vorsehen, entsteht die Forderung, dass die maximale Abweichung zweier beliebiger Komponentenuhren über die komplette Betrachtungszeit hinweg stets kürzer sein muss als die halbe Schrittweite.

5.5.3 Schrittweiten-Modell-Interdependenz

Neben der durch Prozessaggregation (Informationsverdichtung) über jedem Zeitschritt hervorgerufenen zeitlichen Unschärfe bringt die Diskretisierung leider auch eine modellseitige Unschärfe mit sich, da je nach gewählter Diskretisierungsschrittweite ein anderes Modell entsteht: Diskretisiert man den kontinuierlichen Automaten aus Abbildung 27 mit $\Delta t = 1$, so wird aus den beiden Diraczeitbedingungen $x_{1_{\texttt{LA}}} = 8{,}5$ und $x_{1_{\texttt{OK}}} = 1{,}5$: $x_{1_{\texttt{LA}}} = 9$ und $x_{1_{\texttt{LA}}} = 2$. Für einen „normalen“ Prozessdurchlauf (`OK, LA`) ergäbe sich somit eine Durchlaufzeit von 11 statt 10 Zeiteinheiten. Ist der Zyklus die systementscheidende Größe, kann diesem Problem durch Verwendung von Differenzen begegnet werden, womit aus den obigen Zeiten $x_{1_{\texttt{LA}}} = 9$ und $x_{1_{\texttt{OK}}} = 10 - 9 = 1$ würde. Allerdings ist auch dies eine Verfälschung, deren Vertretbarkeit geprüft werden muss. Besonderes Interesse sollte diesem Punkt auch deshalb gewährt werden, weil dies i.d.R. direkten Einfluss auf die durch $\mathbf{x_i^0}$ vorzugebende Verteilung der Uhranfangswerte hat.

Wie Abbildung 31 illustriert, werden die Auswirkungen auf das Systemverhalten noch größer, wenn man es mit Verteilungen zu tun hat. Während die Verwendung einer Zeitschrittweite von $\Delta t = 0{,}25$ (bei einer Abtastung ab $t = 0$) zu zwei Impulsen mit einem Abstand von 2 Zeittakten führt, verschwindet diese Unterscheidung bei einer Schrittweite von $\Delta t = 0{,}5$ in einem einzigen Impuls. In beiden Fällen wurde jedoch bereits verschleiert, dass das System erst nach einer gewissen Zeitspanne einen Ausgang erzeugt, eine Information, die man erst bei einer Schrittweite von $\Delta t = 0{,}125$ extrahieren könnte.

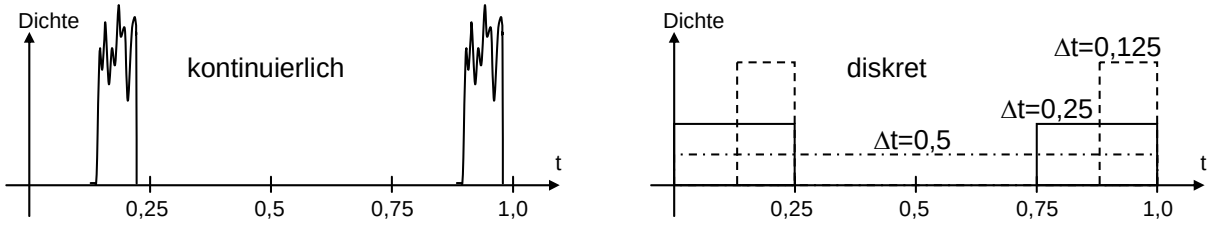

Abbildung 31: Die Schrittweite prägt das zeitliche Verhalten

5.5.4 Übergang vom kontinuierlichen zum diskreten Automaten

Für die Transformation vom kontinuierlichen zum diskreten Automaten ist es wichtig zu beachten, dass nicht alle Bestandteile in der selben Art und Weise definiert worden sind. Diese Diskrepanz ist darauf zurückzuführen, dass der diskrete Automat so definiert wurde, dass er möglichst nahe am zu erzeugenden PRISM-Code ist. Dagegen wurde der kontinuierliche Automat derart definiert, dass sich damit Struktur und Eigenschaften von NAS möglichst gut wiedergeben lassen. Immerhin umfassen (trotz dieser unterschiedlichen Definitionsmotivation) die Mehrzahl der Komponenten analoge Elemente: $\mathbf{Z_{p_i}} = \mathtt{Z_i}$, $\mathbf{W_i} = \mathtt{W_i}$ und $\mathbf{V_i} = \mathtt{V_i}$ sowie $\mathbf{X_i} = \mathtt{X_i}$, falls man davon absieht, dass die Uhren in $\mathtt{X_i}$ über $\mathbb{R}_0^+$ und in $\mathbf{X_i}$ über $\mathbb{N}_0$ definiert sind.

Die Ursache für die Unterschiedlichkeit der verbleibenden Mengen und Funktionen ist, dass im kontinuierlichen Modell strikt zwischen bedingungsbasierten Ereignissen und Zei-

ten unterschieden wird, während dies bei PRISM nicht getan wird. Dies bedeutet, dass die Zustandsübergangsrelation $\mathtt{L_i}$ des kontinuierlichen Modells zur Bildung von $\mathbf{Q_i}, \mathbf{c_i}, \mathbf{p_i}$ des diskreten Modells ausgewertet und hierbei auch die Menge $\mathbf{Z_{t_i}}$ der transienten Zustände erzeugt werden muss. Für jeden abgehenden Übergang eines jeden Zustandes $\mathtt{Z_i}$ des kontinuierlichen Modells muss wie folgt verfahren werden:

1. Initialisierung der Uhr des Zustandes.
2. Falls $\mathtt{d}_{\mathtt{i}_{\mathtt{k_j}}} = \diamondsuit$ ist, entspricht die diskrete Bedingung $\mathbf{c}_{\mathbf{i}_{\mathbf{k_j}}}$ der kontinuierlichen, während die zugehörige Wahrscheinlichkeit $\mathbf{p}_{\mathbf{i}_{\mathbf{k_j}}}$ zunächst gleich eins gesetzt wird.
3. Andernfalls muss für alle diskreten Zeitschritte, für welche das Integral von $\mathtt{d}^*_{\mathtt{i}_{\mathtt{k_j}}}(\mathtt{x})$ über den zugehörigen Zeitbereich[4] $r \cdot \Delta t$

$$\mathbf{p}_{\mathbf{i}_{\mathbf{k_j}}}(\mathbf{x} = r) = \begin{cases} \int\limits_{r\cdot\Delta t+\varepsilon\cdot h(r-1)}^{(r+1)\cdot\Delta t} \mathtt{d}^*_{\mathtt{i}_{\mathtt{k_j}}}(\tilde{\mathtt{x}})\ d\tilde{\mathtt{x}} & \text{für } r \leq 2 \\ \int\limits_{r\cdot\Delta t+\varepsilon}^{(r+1)\cdot\Delta t} \mathtt{d}^*_{\mathtt{i}_{\mathtt{k_j}}}(\tilde{\mathtt{x}})\ d\tilde{\mathtt{x}} \ \cdot\ \left(\prod_{s=1}^{r-1} \mathbf{p}_{\mathbf{i}_{\mathbf{k_j}}}(\mathbf{x} = s)\right)^{-1} & \text{für } r > 2 \end{cases} \tag{5.20}$$

   größer als Null ist, ein Zustandsübergang von diesem persistenten zu einem transienten Zustand gebildet werden. Die zugehörige diskrete Schaltbedingung $\mathbf{c}_{\mathbf{i}_{\mathbf{k_j}}}$ entspricht hierbei der Bedingung aus dem kontinuierlichen Modell, UND-verknüpft mit dem Wert der diskreten Uhrvariablen: $\mathbf{c}_{\mathbf{i}_{\mathbf{k_j}}} = \mathtt{c}_{\mathtt{i}_{\mathtt{k_j}}} \wedge (\mathbf{x} = r)$. Handelt es sich bei $\mathtt{d}^*_{\mathtt{i}_{\mathtt{k_j}}}(\mathtt{x})$ um einen Dirac-Impuls, so kann direkt mit Punkt 4 weiter verfahren werden. Andernfalls muss der stochastisch bedingte Wiedereintritt implementiert werden (vgl. Abbildung 32). Dies bedeutet, dass der zu erzeugende transiente Zustand zwei abgehende Übergänge aufweist: Der erste der beiden verbindet diesen transienten Zustand mit der durch den Zustandsübergang des kontinuierlichen Modells gegebenen Menge nachfolgender persistenter Zustände (siehe viertens). Die zugehörige Aktivierungswahrscheinlichkeit wird aufgrund von Gleichung 5.20 bestimmt. Hierbei ist zu beachten, dass – wie bereits für die serielle Wertzuweisung in Abschnitt 4.6.2 erläutert – die jeweilige Wahrscheinlichkeit auf den noch verbliebenen Anteil normiert werden muss. Aus den Wahrscheinlichkeiten $\left[\frac{1}{4};\ \frac{1}{3};\ \frac{5}{12}\right]$ wird also $\left[\left(\frac{1}{4}, \frac{3}{4}\right);\ \left(\frac{4}{9}, \frac{5}{9}\right);\ (1)\right]$. Die Uhrmodifikation wird aus dem kontinuierlichen Modell übernommen (s.u.).
   Die zweite der beiden Zustandsübergänge verbindet den aktuellen transienten mit dem vorhergehenden persistenten Zustand (Selbstschleife).
4. Wenn der durch die Übergangsrelation $\mathtt{L_i}$ gegebene Wert für den betrachteten Übergang nicht identisch zu eins ist, bedeutet dies, dass dieser Übergang nicht in jedem Fall, sondern nur mit einer gewissen Wahrscheinlichkeit aktiviert wird. Folgerichtig müssen auch die nach Gleichung 5.20 bestimmten Werte ($\mathbf{p}_{\mathbf{i}_{\mathbf{k_j}}}$) mit dieser Wahrscheinlichkeit multipliziert werden.

[4] Hinweis: Die Zeitintervalle sind zu $x \in (r \cdot \Delta t,\ (r+1) \cdot \Delta t]$ definiert, da die Uhren auf den Wert 0 zurückgesetzt werden und von dort mit dem Zählen beginnen. Aus der kontinuierlichen Bedingung $x = 3$ wird somit $x + \Delta t \geq \frac{3}{\Delta t}$ bzw. für $\Delta t = 1$: $x = 2$.

5. Eine Uhr $\mathbf{x_{i_g}}$, die im Zustand $\mathbf{z_{i_k}}$ aktiv war und die beim Eintritt in den nachfolgenden Zustand keinen Reset erfährt, wird um eins erhöht (inc in $\mathbf{f_i}$, vgl. z.B. Übergang von `LA` nach `BS` in Abbildung 29).
6. Die Zuordnung der Ausgangsvariablen ergibt sich direkt aus dem kontinuierlichen Automatenmodell.

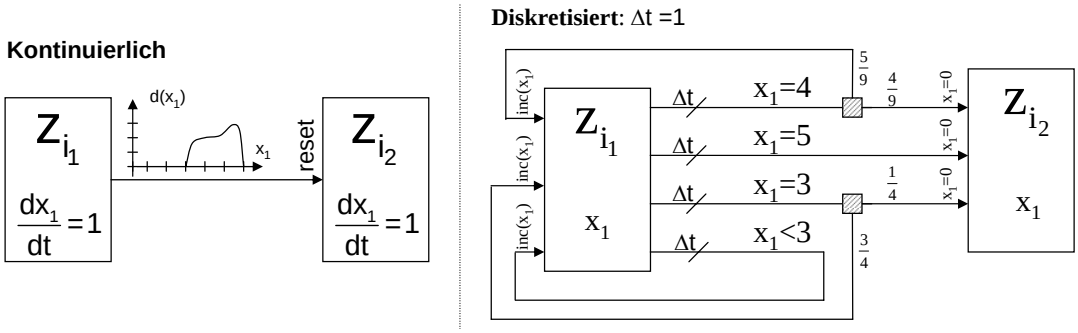

Abbildung 32: Zustandsübergang mit zeitverteiltem Auslösezeitpunkt (Typ (2)) in zeitkontinuierlicher und -diskreter Darstellung

Für jeden entstandenen persistenten Zustand $\mathbf{z_{i_k}}$ müssen darüber hinaus die folgenden fünf Schritte abgearbeitet werden:

1. Sollten zwei oder mehr von einem transienten Zustand abgehende Übergänge gleiches Folgeverhalten besitzen, werden diese zusammengefasst (Abbildung 33).
2. Folgen auf einen persistenten Zustand mehrere, identisches Folgeverhalten aufweisende transiente Zustände, so sollten die Bedingungen der zugehörigen Übergänge zusammengefasst und die Übergänge durch einen einzigen Übergang ersetzt werden (Abbildung 33).

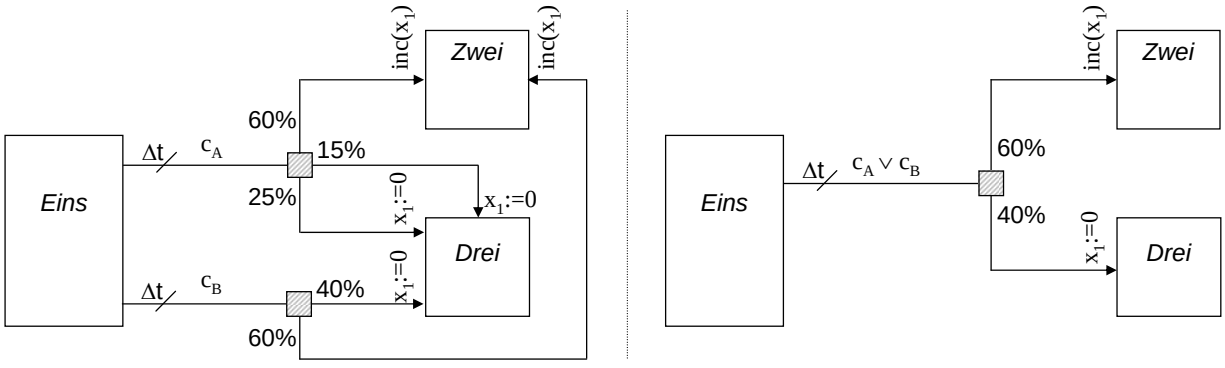

Abbildung 33: Zusammenfassen von Übergängen mit gleichem Folgeverhalten

3. Eine Selbstschleife muss initialisiert werden, die genau dann ermöglicht wird, wenn sonst kein anderer abgehender Übergang ermöglicht ist (Gleichung 5.18).
4. Die Uhrmodifikationen aller von diesem Zustand mittelbar abgehenden Zustandsübergänge werden entsprechend der Einträge in $\mathtt{X_i^R}$ und $\mathtt{X_i^{act}}$ bestimmt: Uhren, die im neuen Zustand aktiv sind, werden beim Eintritt um eins erhöht $\{inc\}$, inaktive

Uhren behalten ihren Wert hingegen bei $\{-\}$, wobei Rücksetzen $\{res\}$ dominant gegenüber der Aktivität ist (d.h. bei Rücksetzung wird durch $\mathbf{f_i}$ ein $\{res\}$ zugewiesen, unabhängig davon, ob $\mathtt{X}_\mathtt{i}^{\mathtt{act}}$ eine aktive oder inaktive Uhr vorsieht). Selbstschleifen sind, so sie nicht bereits im kontinuierlichen Modell vorhanden waren, von jeglicher Rücksetzung ausgenommen, können also nur im Wert belassen $\{-\}$ oder eine Werterhöhung um eins erfahren $\{inc\}$.

5. Weist ein Zustand mehrere abgehende Übergänge auf, deren Bedingungen nicht identisch sind, jedoch im gleichen Zeitschritt erfüllt werden können, muss dies – wie bereits auf Seite 83 diskutiert – entweder durch eine paarweise disjunkte Priorisierung verhindert oder durch Einführung eines zusätzlichen Übergangs explizit vorgesehen werden.

Abschließend müssen noch die Anfangsverteilungen der Zustände $\mathtt{Pr}_\mathtt{i}^0$ und Uhren $\mathtt{X}_\mathtt{i}^0$ des kontinuierlichen Modells auf die diskreten Anfangsverteilungen $\mathbf{z}_{\mathbf{p_i}}^{\mathbf{0}}$ und $\mathbf{x}_\mathbf{i}^\mathbf{0}$ transformiert werden. Prinzipiell handelt es sich beim Übergang von $\mathtt{X}_\mathtt{i}^0$ auf $\mathbf{x}_\mathbf{i}^\mathbf{0}$ lediglich um eine einfache vom Kontinuierlichen ins Diskrete weisende Abbildung. Da die Zeit im Diskreten jedoch nicht mehr kontinuierlich ist, müssen die Anfangszeit-Dichteverteilungen eines jeden diskreten [Zustand, Zeit]-Paares durch Integration des kontinuierlichen [Zustand, Zeit]-Paares über jedweden diskreten Zeitschritt bestimmt werden. In anderen Worten werden hierbei die Dichtefunktionen des Kontinuierlichen in eine Menge diskreter Wahrscheinlichkeiten überführt. Abbildung 34 zeigt dies beispielhaft für zwei typische Anfangsverteilungen, namentlich der Gleichverteilung (links) und der Exponentialverteilung (rechts). Gleichverteilungen treten z.B. dann auf, wenn die Aufenthaltsdauer durch eine konstante Zeitspanne (Übergangstyp (1)) gegeben ist und alle Eintrittszeitpunkte in diesen Zustand gleichwahrscheinlich sind. Exponentialverteilungen ergeben sich hingegen z.B. dann, wenn das Verhältnis der Wahrscheinlichkeiten für das Verweilen im Zustand zum Zustandswechsel für alle Zeitintervalle stets gleich groß ist ($d(x) = \lambda \cdot e^{-\lambda x}$). Ein solches Verhalten findet sich beispielsweise im Zustand `untätig` des in Abbildung 12 dargestellten Maschinenprozesses.

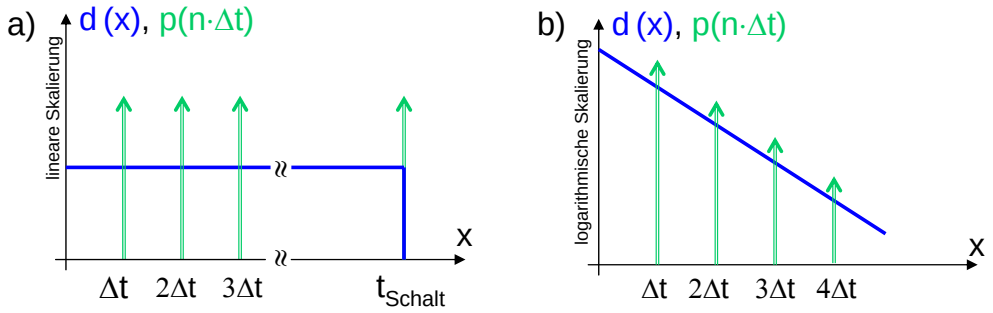

Abbildung 34: Überführung der Dichtefunktionen des Kontinuierlichen in eine Menge diskreter Wahrscheinlichkeiten

Die Transformation der Anfangsverteilungen der Zustände gestaltet sich hingegen im allgemeinen Fall als unbestimmt, da die Transformation der sich an den Übergängen befindlichen Zeitbedingungen nicht zwingend urbildtreu ist (vgl. Abschnitt 5.5.3). Nur wenn

letzteres der Fall ist, kann $\mathbf{z}^{\mathbf{0}}_{\mathbf{p_i}}$ direkt aus $\mathtt{Pr}^0_{\mathtt{i}}$ bestimmt werden: diese sind dann identisch. Im Normalfall wird es jedoch unumgänglich sein, $\mathbf{z}^{\mathbf{0}}_{\mathbf{p_i}}$ aufgrund des durch die diskreten Zeitbedingungen gegebenen diskreten Eigenverhaltens neu zu bestimmen.

5.6 Transformation nach PRISM

Die Transformation des diskreten Automaten in die PRISM-Modellsprache gestaltet sich schlussendlich als direkte Umsetzung. Für jeden Teilautomaten muss hierzu ein Modul in der in Abbildung 35 beispielhaft gezeigten Form generiert werden.

```
module <modname>
  <statevar>:    [<range>] init <initial_s>;
  <countervar>:  [<range>] init <initial_c>;
  [Dt]  <ck1>& <zk>    ->  <pk11>:(<zk+1,1,1>)&(<fk11>)
                             + <pk12>:(<zk+1,1,2>)&(<fk12>)
                             + ...
                             + <pk1h>:( <zk+1,1,h>)&(<fk1h>);
  [Dt]  <ck2>& <zk>    ->  <pk12>:(<zk+1,2,1>)&(<fk21>) + ...;
   ⋮          ⋮            ⋮                  ⋮
  [Dt]  <ckm>& <zk>    ->  <pkm1>:(<zk+1,m,1>)&(<fkm1>) + ...;
   ⋮          ⋮            ⋮
  [Dt]  <ckm+1>& <zk>  ->  <zk>;
endmodule
```

Abbildung 35: Generischer PRISM-Code für den Teilautomaten <modname>, wobei nur die Befehlszeilen des [Zustand, Zeit]-Paars <zk> dargestellt wurden.

<modname> repräsentiert hierbei den Namen des Moduls, ist also ein Bezeichner für den gerade zu implementierenden Teilautomaten. <statevar> bezeichnet eine Integer-Variable, in der gespeichert wird, in welchem Zustand sich der Automat befindet. Das folgende <range> gibt den Wertebereich von <statevar> an und somit die maximale Anzahl an Zuständen (dies entspricht der Mächtigkeit von $\mathbf{Z}_{\mathbf{p_i}}$). <initial_s> wird verwendet, um einen Anfangszustand für <statevar> auszuwählen. Analoges gilt für <countervar>, mit dem Unterschied, dass es sich hierbei um die Uhrvariable des Automaten handelt und in <range> somit der maximal und minimal auftretende Uhrenwert gespeichert wird. Besitzt ein Teilautomat mehr als eine Uhr, muss der Code entsprechend erweitert werden. [Dt] ist ein Synchronisationszeichen, das besagt, dass ein Modul eine mit [Dt] bezeichnete Zeile nur dann schalten darf, wenn in allen Modulen, die eine mit [Dt] beginnende Codezeile haben, auch jeweils (wegen Gleichung 5.16 sogar genau) eine [Dt]-Codezeile eine erfüllte Schaltbedingung aufzuweisen hat. Dies ist gemäß Gleichung 5.17 gewährleistet, wobei aufgrund der einheitlichen Zeitbasis alle normalen Anweisungszeilen mit [Dt] beginnen. Ausnahmen hiervon sind lediglich die Anweisungszeilen eines für die Ermittlung des Anfangszustands notwendigen Vorprozesses sowie diejenigen des Terminierungsschritts (vgl. Kapitel 4).

<ck1> repräsentiert die Bedingung $\mathbf{c}_{\mathbf{i}_{\mathbf{k}_1}}$ des Zustands $\mathbf{z}_{\mathbf{i}_\mathbf{k}}$ und <zk> ist die Kurzform für <statevar> = k & <countervar> = q. Letzteres ist eine boolesche Bedingung dafür, dass sich der Teilautomat sowohl im k. Zustand befindet als auch die interne Uhr den Wert q aufweist. „->“ symbolisiert das Ausführen des Übergangs und <pk11> entspricht $\mathbf{p}_{\mathbf{i}_{\mathbf{k}_{1_1}}}$ (erster Arm der stoch. Verzweigung des ersten vom Zustand $\mathbf{z}_{\mathbf{i}_\mathbf{k}}$ abgehenden Übergangs).

<zk+1,1,1> bezeichnet jenes [Zustand, Zeit]-Paar, das den Zustand des Automaten nach dem Ausführen des Übergangs beschreibt. Dies kann ein anderer oder erneut der gleiche Zustand (dann jedoch mit i.d.R. um eins erhöhter Zeitvariablen) sein. <fk,1,1> schlussendlich bezeichnet die Uhrmodifikationsfunktion $\mathbf{f}_{\mathbf{i}_{\mathbf{k}_{1_1}}}$.

Abbildung 36 zeigt den PRISM-Code für das in Abbildung 29 gezeigte Beispiel. Die Zustände sind mit 1 für LA, 2 für BS, 3 für OK und 4 für FE kodiert worden. Die Maximalwerte der beiden Uhren x_1 und x_2 ergeben sich aufgrund der Übergangsbedingungen zu 8 sowie 4. x_2 hat zu Beginn den Wert 0, wohingegen die Anfangswerte von x_1 und Zst in einem Vorprozess zugewiesen werden. Der Nachprozess weist den Variablen einen Terminalwert zu.

```
module Beispiel29
  Zst:  [0..4] init 0; // 1: LA, 2: BS, 3: OK, 4: FE
  x1 :  [0..8] init 0;
  x2 :  [0..4] init 0;
  [pre]  Zst=0                        ->  1/11:(Zst'=1)&(x1'=0) +
                      1/11:(Zst'=1)&(x1'=1) + ... + 1/11:(Zst'=1)&(x1'=8) +
                      1/11:(Zst'=3)&(x1'=0) + 1/11:(Zst'=3)&(x1'=1);
  [Dt]   Zst=1 & IRQ & x1<8       ->  x1'=x1+1 & x2'=0 & Zst'=2;
  [Dt]   Zst=1 & !IRQ & x1<8      ->  x1'=x1+1;
  [Dt]   Zst=1 & x1=8             ->  0.99:(Zst'=3)&(x1'=0) +
                                      0.01:(Zst'=4)&(x1'=0);
  [Dt]   Zst=2 & x2<2             ->  x2'=x2+1;
  [Dt]   Zst=2 & x2=2             ->  3/4:(x2'=x2+1) + 1/4:(Zst'=1)&(x2'=0);
  [Dt]   Zst=2 & x2=3             ->  5/9:(x2'=x2+1) + 4/9:(Zst'=1)&(x2'=0);
  [Dt]   Zst=2 & x2=4             ->  Zst'=1&x2'=0;
  [Dt]   Zst=3 & ( x1=1 | Abbruch )  ->  x1'=0 & Zst'=1;
  [Dt]   Zst=3 &  x1<1 & !Abbruch  ->  x1'=x1+1;
  [Dt]   Zst=4 &  Reset           ->  x1'=0 & Zst'=1;
  [Dt]   Zst=4 &  !Reset          ->  true;
  [post] true                     ->  x2'=0 & x1'=0 & Zst'=1;
endmodule
```

Abbildung 36: PRISM-Code des Beispiels aus Abbildung 29, wobei für den Zustand BS das in Abbildung 32 dargestellte Ausgangsverhalten verwendet wurde.

Anmerkung: Der von PRISM benötigte Speicherplatz lässt sich in geringem Umfang dadurch reduzieren, dass alle auftretenden Zustands-Uhr-Kombinationen in eigenständige Zustände umgewandelt und in Folge nur noch Zustände und keine Uhren mehr im System

vorkommen. Im Beispiel kann hierdurch eine Reduktion des Zustandsraums des Modules `Beispiel29` von 180 auf 58 Zustände erreicht werden. Dies allerdings ist ein Programmierdetail und hat wenig mit dem Entwurf eines „allgemeinen Modellierungsansatzes“ zu tun.

5.7 Multischrittweitensteuerung

Bisher wurde ein Modellierungskonzept verwendet, bei welchem die Zeit in immer gleicher (diskreter) Weise voranschreitet. Für Systeme jedoch, bei denen sich nur zu wenigen Zeitpunkten eine Änderung ergibt, ist dies wenig ressourcenschonend. Aus diesem Grunde wurde ein Ansatz entwickelt, der jene Zeitpunkte überspringt, zu denen sich garantiert kein Zustandswechsel ergibt. Vorraussetzung für einen solchen Mechanismus ist jedoch zum einen eine unterschiedlich lange Schrittweiten unterstützende Automatenstruktur und zum anderen ein Schätzalgorithmus, welcher die notwendige Länge $\theta \cdot \Delta t$ des nächsten Schrittes bestimmt.

Wie Abbildung 37 zeigt, ist die Umsetzung vom Kontinuierlichen ins Diskrete auch für den Fall einer variablen Schrittweite mit dem in Abschnitt 5.4 vorgeschlagenen Automaten möglich, falls das Verhalten des inc(x) Operators entsprechend angepasst wird. Letzterer erhöht dann x nicht mehr um einen einzigen Zeitschritt (Δt), sondern um die durch $\theta \cdot \Delta t$ gegebene Zeitspanne. Die notwendigen Übergangsbedingungen ergeben sich analog.

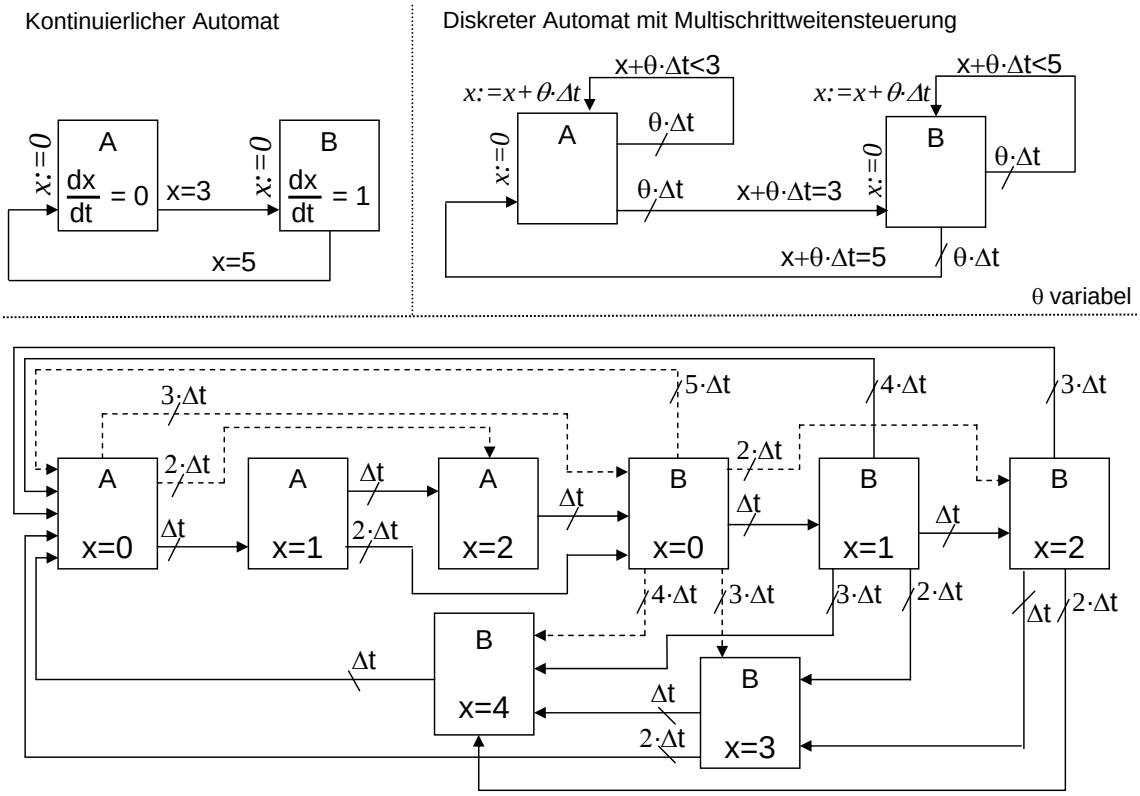

Abbildung 37: Kontinuierlicher Automat (oben links) und zugehörige multischrittgesteuerte diskrete Darstellung (oben rechts und unten)

Aufgrund des modularen Charakters der verwendeten Automatenstruktur ergibt sich hierdurch der folgende Perspektivenwechsel: Nicht mehr das auf ein Ereignis reagierende – und infolge einen Zustandswechsel ausführende – sondern das dieses Ereignis erzeugende Modul übernimmt die Initiative. Dies bedeutet, dass jeder Teilautomat aufgrund der momentanen Gesamtsituation eine Schätzung darüber abgeben muss, in welcher zeitlichen Entfernung er ein für andere Teilautomaten möglicherweise interessantes Ereignis erzeugt oder er sein Verhalten grundlegend ändert.

Um dies zu erläutern wurde der in Abbildung 37, rechts oben dargestellte Automat in der im unteren Teil der Abbildung gezeigten Form wiedergegeben. Hierbei wurde für jede mögliche (diskrete) Kombination aus Zeitvariable x und Zustand ein eigener Zustand geschaffen und die sich direkt ergebenen Übergänge eingezeichnet. Die Funktion ist dabei kurz erklärt die Folgende: Befindet sich der Automat im Zustand A mit Zeit x = 0, so kann er entweder die Zeit um ein, um zwei oder um drei mal Δt erhöhen. Drei ist hierbei das Maximum, da nach dieser Zeit ein Zustandswechsel nach B stattfindet. Gleiches gilt für Zustand B zum Zeitpunkt x = 2.

Aus dieser Darstellung lässt sich demnach für jeden Zustand der maximal zulässige Wert von θ ablesen, welcher als $\theta_{i_{\breve{r}}}$ bezeichnet sei. Tatsächlich ausgeführt wird jedoch die durch $\theta_{\breve{r}}$ gegebene Schrittanzahl, welche über das Minimum der $\theta_{i_{\breve{r}}}$ aller Teilautomaten definiert ist:

$$\theta_{\breve{r}} = \min_i \theta_{i_{\breve{r}}} \tag{5.21}$$

r steht hierbei für den r-ten Teilschritt und $\breve{}$ zeigt an, dass es sich um einen Schätzwert der maximal zulässigen Schrittweite bis zum nächsten für einen der anderen Teilautomaten möglicherweise interessanten Zustand handelt. In diesem „für einen der anderen Teilautomaten möglicherweise interessanten Zustand“ liegt hierbei die eigentliche Problematik: Der Teilautomat kann aufgrund des modularen Charakters des Gesamtsystems gar nicht wissen, ob und vor allem wann diese Information für einen anderen Automaten von Interesse ist. Hierzu ein Beispiel: Angenommen, der Automat wäre gerade in den Zustand (`B`, x=0) gewechselt und das damit einhergehende Verlassen eines mit A markierten Zustandes habe das – im Automaten des Maschinenbeispiels aus Abbildung 20 verwendete – Ereignis *Werkstück* erzeugt. Befindet sich der Maschinenprozess im Zustand `Bearbeiten`, so spielt dies keine weitere Rolle: Im Automat aus Abbildung 37, unten, könnte man sich in diesem Fall mit bis zu fünffacher Schrittlänge voranbewegen. Befindet sich der Maschinenprozess jedoch im Zustand `Bereit`, so benötigt dieser Automat die mit dem Ereignis einhergehende Information umgehend, was der minimal möglichen Schrittweite ($\theta = 1$) entspricht. In der Folge werden also mehr Ereignisse berücksichtigt, als minimal erforderlich sind.

Die Gesamtsystemsmodularität erfordert demnach die Verwendung konservativer Schätzungen, was im ungünstigsten Fall ein Fortschreiten mit minimaler Geschwindigkeit bedeutet ($\theta = 1$). Im betrachteten Beispiel der Abbildung 37 führt dies dazu, dass die gestrichelt gezeichneten von den beiden (x=0)-Zuständen abgehenden Übergänge entfallen. Andererseits

ermöglicht die Gesamtsystemsmodularität erst eine effiziente Implementierung des Schätzalgorithmus. Dies liegt u.a. daran, dass der Schätzalgorithmus als Zuweisung modelliert werden muss, was zur Folge hat, dass mit der Zuweisung des neuen Zustandes auch gleichzeitig der Wert von $\theta_{i_{\check{r}}}$ ermittelt wird. Die hierfür notwendige Information, welche Werte θ im gerade eingenommenen Zustand annehmen kann, kann jedoch nur durch Auswertung der von diesem Zustand abgehenden Übergänge ermittelt werden. Damit der Schätzalgorithmus hierfür lediglich das Verhalten des eigenen Teilautomaten berücksichtigen muss, also davon ausgehen darf, dass sich der Zustand der anderen Teilautomaten nicht geändert hat, wurde zuvor festgelegt, dass im aktuellen Zeitschritt kein externes Ereignis erzeugt worden sein kann, ohne dass $\theta_{\check{r}}$ vom dieses Ereignis erzeugt habenden Teilautomaten auf eins gesetzt worden wäre.

Fazit: Während sich durch Verwendung der Multischrittweitensteuerung der entstehende tatsächliche Automatenzustandsraum deutlich verringern lässt, erhöht sich – durch die zur Schrittweitensteuerung notwendigen zusätzlichen Variablen – die theoretische Zustandsraumgröße und somit die Zeit, welche notwendig ist, um das Automatenzustandsraummodell zu generieren (model building time). Dieser Ansatz macht daher insbesondere dann Sinn, wenn die Anzahl der Ereignisse bzw. der relevanten Module gering ist. Für ein ausführliches Beispiel einer erfolgreichen Implementierung wird auf [Greifeneder und Frey, 2006c] verwiesen.

Anmerkungen zur Implementierung:

1. Die Minimumsbildung kann in PRISM als `formula` außerhalb der Modulblöcke definiert werden.

2. Ergibt die Schätzung einen reellwertigen Wert, so ist aus Konvergenzgründen der nächsthöhere ganzzahlige Wert zu verwenden.

3. Da die Schrittweite mit $\theta \cdot \Delta t$ nicht mehr konstant ist, kann aufgrund der Anzahl der Übergänge nicht mehr auf die tatsächliche Zeitspanne geschlossen werden. Ein pfadbasierter Ansatz ist daher nur dann möglich, wenn die einzelnen Teilpfade entsprechend ihrer zeitlichen Länge gewichtet werden.

4. Weist ein Teilautomat einen finalen Zustand oder Zustände mit nur ereignisbasierten Bedingungen auf, so könnte er in diesen Zuständen sein $\theta_{i_{\check{r}}}$ zu ∞ oder auf einen beliebig großen Wert bestimmen. Während ersteres in PRISM nicht möglich ist, erzeugt zweiteres einen unnötig großen Zustandsraum. Aus diesem Grunde ist es in diesen Fällen sinnvoll, die Schätzung von $\theta_{i_{\check{r}}}$ um eine boolesche Variable zu ergänzen, die anzeigt, ob dieser Wert in die Minimumsbildung einfließen soll oder nicht, womit der Definitionsbereich von $\theta_{i_{\check{r}}}$ relativ eng gewählt werden kann.

6 Anwendungsbeispiel

Bereits in Abbildung 1 wurde die Struktur eines typischen NAS vorgestellt. Dieses soll nun als Anwendungsbeispiel Verwendung finden. Die zu untersuchende Eigenschaft ist natürlich die Antwortzeitverteilung des Systems, wobei auch diskutiert werden soll, welchen Anteil die unterschiedlichen NAS-Verhaltenseigenschaften Bearbeitung, Netzübertragung, Synchronisation zyklischer Prozesse, Zugriffskonflikte und Fehler an der mittleren Antwortzeit haben.

In Abschnitt 6.1 wird zunächst das Verhalten eines aus einer SPS, der zugehörigen SPS-I/O, der Anlagen-I/O, dem Netzwerk, einem Sensor mit Signal-Typ I Verhalten und einem Stellglied bestehenden Grundsystems betrachtet, wobei alle auftretenden Zeiten als konstant angenommen werden.

Abschnitt 6.2 erweitert dies dann von konstanten Zeiten auf die Verwendung von Zeitverteilungen für den SPS-Zyklus einerseits und die Netzübertragung andererseits. Ebenfalls eine Erweiterung des Grundmodells wird in Abschnitt 6.3 vorgenommen, welcher die Auswirkungen von Zugriffskonflikten und Fehlern auf die Antwortzeitverteilung thematisiert.

Die vorgestellte Analysemethodik ermöglicht die Untersuchung vielfältiger Fragestellungen. Beispielhaft wird in Abschnitt 6.4 eine Anwendung vorgestellt, bei der aufgrund von Antwortzeitverteilungen unterschiedliche Systemkonfigurationen bewertet werden. Ein zweites Beispiel demonstriert die Bestimmung der minimalen Haltezeit, wie sie bei einem Signal-Typ II auftritt. Den Abschluss des Kapitels bildet Abschnitt 6.5, in welchem die mittels PMC erhaltenen analytischen Werte mit dem Ergebniss umfangreicher Messungen verglichen werden.

6.1 Analyse des Komponentenverhaltens

Das Zeitverhalten des zu diskutierenden Grundsystems ergibt sich aus dem bereits in Abbildung 2 dargestellten GANTT-Chart und entspricht in der Funktionalität einem Schneider SPS-System mit MODBusIP [Denis et al., 2007]: Nachdem sich der Sensorwert ändert, wartet die Anlagen-I/O zunächst auf eine Anfrage der SPS-I/O. Trifft diese ein, erstellt die Anlagen-I/O ein Antworttelegramm. Dieses geht über das Netz zur SPS-I/O und wird von dort über den Backplane-Bus direkt in den Eingangspuffer der SPS geschrieben. Nun wird auf den nächsten Lesevorgang der SPS gewartet. Nach dem Lesen wird der Wert in der SPS verarbeitet und eine Reaktion berechnet. Diese geht wieder zur SPS-I/O und wartet dort auf den nächsten Sendevorgang. Danach erfolgt ein erneuter Netzdurchgang und eine Bearbeitung des Telegramms in der Anlagen-I/O. Hierbei ist zu beachten, dass der Zeitpunkt des Signalwechsels am Sensor unabhängig vom restlichen Prozess ist und bis kurz vor dem Eintreffen der Anfrage an der Anlagen-I/O erfolgen kann, um noch für diese Anfrage berücksichtigt zu werden.

6.1.1 Grundmodule

Für den zu modellierenden Prozess werden – neben dem bereits in Abschnitt 5.2 eingeführten SPS-Modul – noch eine Reihe weiterer Module benötigt. Die zusätzlich benötigten DesLaNAS-Einzelmodule werden im Folgenden vorgestellt. Hierbei ist anzumerken, dass zwar in jedem Modul eine mit x bezeichnete lokale Zeitvariable vorkommt, die einzelnen (lokalen) Zeitvariablen aber unabhängig voneinander und nur über das Fortschreiten der globalen Zeit (t) gekoppelt sind.

Netzübertragung

Die Netzübertragung wird, wie im linken Teil der Abbildung 38 dargestellt, modelliert. Dieser Automat verharrt im mit `untätig` bezeichneten Zustand, ohne dass die lokale Uhr läuft. Sobald ein zu übermittelndes Paket eintrifft (Ereignis senden), wechselt der Automat in den mit `befördern` bezeichneten Zustand. Die Übertragungszeit selbst wird durch die Verteilungsfunktion $\mathrm{d_{net}}(\mathtt{x})$ – d.h. als Wahrscheinlichkeitsverteilung über der Zeitachse – gegeben. Für die hier durchgeführte Betrachtung wird die Netzübertragungszeit als jene Zeit definiert, die sich als Summe aus den Einzelprozessen einer Netzübertragung ergibt, wie z.B. Übertragen, Verpacken, Entpacken, Switching.

Die in der Automatisierungstechnik eingesetzten Netzwerkkomponenten sind in der Regel sehr schnell (Switching times von weniger als 0,08 ms) und selten überlastet [Denis et al., 2007]. Aus diesem Grunde ist es zulässig jede der zu modellierenden Datenübertragungen als eigenständiges (d.h., von den anderen Datenübertragungen unabhängiges) Modul zu instanziieren und Störeinflüsse wie Verlust, Stau und andere Übertragungsverzögerungen als getrennte Prozesse zu modellieren (vgl. hierzu Abschnitt 6.3.1). Folglich erfordert das zu modellierende System zwei Netzmodule, namentlich eines für den Hinweg von der SPS-I/O zur Anlagen-I/O und eines für den Rückweg. Vereinfachend wird zunächst jede Netzpassage mit konstant 2 ms berücksichtigt, was nach [Gottheit, 2006] dem Mittelwert einer LAN-Übertragung entspricht. Durch Diskretisierung mit $\Delta t = 1$ ms ensteht somit der auf der rechten Seite von Abbildung 38 dargestellte diskretisierte Automat. Hieraus lässt sich der PRISM-Code wie in Abbildung 39 gezeigt, erstellen.

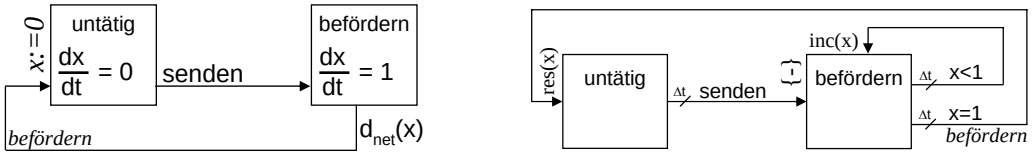

Abbildung 38: Netzmodul in zeitkontinuierlicher (links) und in zeitdiskreter Ausführung (rechts). Die Diskretisierung erfolgte für eine konstante Netzverzögerung von 2 ms und einer Diskretisierungsschrittweite von $\Delta t = 1$ ms.

Der Zustand `untätig` wurde mit 0 und der Zustand `befördern` mit 1 kodiert. Die zugehörige Zustandsvariable lautet `Net1Zst`. Die Uhrvariable `x` wurde `Net1x` benannt und das von extern aufzunehmende Ereignis `SendenNet1`. Letzteres wird im Zuge der Instanziierung entsprechend ersetzt, z.B. durch `SPSIO1Anfrage`, wenn auf das Ereignis $Anfrage$ verlinkt

```
module Net1
  Net1Zst:  [0..1] init 0; // 0: untätig, 1: befördern
  Net1x :   [0..1] init 0;
  [Dt]    Net1Zst=0 & !SendenNet1  ->  Net1Zst'=0 & Net1x=0 ;
  [Dt]    Net1Zst=0 & SendenNet1   ->  Net1Zst'=1 & Net1x=0 ;
  [Dt]    Net1Zst=1 & Net1x=0      ->  Net1Zst'=1 & Net1x=1 ;
  [Dt]    Net1Zst=1 & Net1x=1      ->  Net1Zst'=0 & Net1x=0 ;
  [post]  true                     ->  Net1Zst'=0 & Net1x=0 ;
endmodule
```

Abbildung 39: PRISM-Code für die Netzübertragung nach Abbildung 38

werden soll, welches vom, auf dem in Abbildung 40 dargestellten Automaten basierenden, Modul SPSIO1 erzeugt wird. Obwohl Ausgangsereignisse als boolesche Variablen in den Code eingebracht werden könnten, ist es unter dem Aspekt der Zustandsraumgröße meistens sinnvoller, diese im PRISM-Header als `formula` zu hinterlegen. Für das Ausgangsereignis *befördern* müsste also `formula Net1befordern = (Net1Zst=1 & Net1x=1);` eingetragen werden, da das Flag gesetzt sein soll, wenn sich der Automat im Zustand `befördern` (`Net1Zst=1`) bei Uhrzählerstand $\mathtt{x} = 1$ (`Net1x=1`) befindet. Auf die mit `[pre]` beginnenden Initialisierungszeilen konnte verzichtet werden, da angenommen wurde, dass das System mindestens zwei Zeitschritte lang Einschwingen kann und sich eine Initialisierung somit von selbst ergibt. Wollte man dies vermeiden, müsste eine explizite Kopplung an den rückzurechnenden Eintrittszeitpunkt des `SendenNet1` Ereignisses implementiert werden. Hierfür müsste allerdings ein zweiter Vorschritt `[pre2]` eingeführt werden, da der erste für die Bestimmung des stochastischen Anfangszustands der Module benötigt wird und daher erst in einem zweiten Vorschritt auf einen stochastisch ermittelten Anfangszustand eines anderen Moduls Bezug genommen werden kann.

Anmerkung: Der in Abbildung 39 gezeigte Code kann noch optimiert werden, da im Zustand `untätig` keine Uhr aktiviert ist und somit der Automat nur drei der vier möglichen Zustandskombinationen einnimmt.

SPS-I/O-Modul

Funktional gesehen hat das SPS-I/O-Modul (Abbildung 40, links) große Ähnlichkeit mit demjenigen der SPS. Die lokale Uhrvariable x wird beim Eintritt in den Zustand `Anfrage` zurückgesetzt. Das Verlassen dieses Zustandes bezeichnet jenen Moment, zu dem die (oder die Menge der) Anfrage(n) abgesendet worden ist. Obwohl diese Anfragen in serieller Reihenfolge versandt werden, wird angenommen, dass dies parallel erfolgt. Diese Annahme ist so lange gültig, wie die Zeitschrittweite des für PMC notwendigen diskreten Modells deutlich größer als die zum Senden einer Anfrage erforderliche Zeit ist und die betrachteten Anfragen zeitlich eng hintereinander gesendet werden. Für das hier vorgestellte System ist es daher ausreichend, einen einzigen Anfrage-Zustand zu verwenden.

Sobald alle Anfragen versandt wurden, wechselt der SPS-I/O-Automat in den `warten` Zustand und verbleibt dort bis zum Ende des Zyklus. Der Eingang der Antworten wird

nicht im SPS-I/O-Automaten abgebildet, da diese direkt in den Backplanebus geschrieben werden und somit der SPS unverzüglich zur Verfügung stehen.

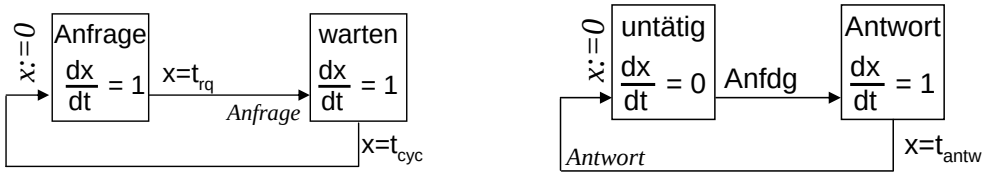

Abbildung 40: SPS-I/O-Modul (links) und Anlagen-I/O-Modul (rechts)

Anlagen-I/O

Mit der Anlagen-I/O ist auch das vierte und letzte der benötigten Module benannt. Dieses ist wie im rechten Teil der Abbildung 40 gezeigt aufgebaut und entspricht von der Struktur demjenigen der Netzübertragung, weist jedoch im Unterschied zu letzterem eine konstante Zeitbedingung auf, da die Bearbeitungszeit der Anlagen-I/O als fix angenommen wurde.

6.1.2 Antwortzeitanalyse

Das zur Bestimmung der Antwortzeitverteilung notwendige Signal-Tracking-Modul ist in Abbildung 41 dargestellt. Es ergibt sich direkt aus dem auf Seite 95 erläuterten Signalpfad und der zu prüfenden Eigenschaft „Verteilung der Antwortzeit“. Die Übergänge wurden hierbei mit den entsprechenden Ereignissen beschriftet. Die Einschwingzeit (Pretime) ist notwendig, um dem durch Netzübertragung und SPS-I/O gegebenen Seitenprozess ausreichend Zeit zu geben, seinen Initialzustand (Zustand zum Ereigniseintritt) zu bestimmen.

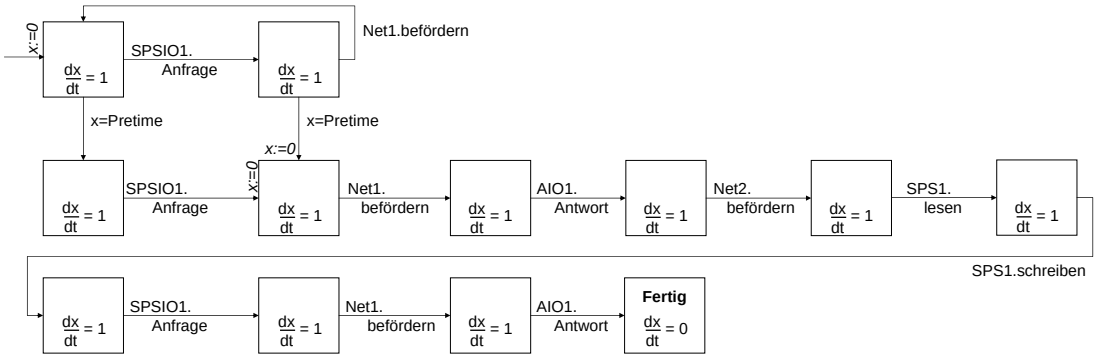

Abbildung 41: Signal-Tracking des Grundbeispiels (ungekürzt)

Da nicht alle entlang des Signalpfades erzeugten Ereignisse auch einen Zustandswechsel des Signal-Tracking-Moduls erforderlich machen, wäre es für das Grundbeispiel sinnvoll, eine Reduktion der Zustandsanzahl vorzunehmen. In einer Folge von Ereignissen A, B und C darf im Signal-Tracking ein Ereignis B dann weggelassen werden, wenn das System keine Möglichkeit vorsieht von A nach C zu gelangen ohne B genau einmal zu passieren (und ohne dass A erneut auftritt). Eine solche Reduktion kann entweder analytisch oder durch entsprechendes Ingenieursgeschick vorgenommen werden.

Die zugehörige Instanziierungs- und Parametrierungssicht des Systems ist in Abbildung 42 gezeigt. Der Aufbau der jeweils ein eigenständiges Modul repräsentierenden Blöcke wurde bereits in Abschnitt 5.2 erläutert. Die Verbindungslinien symbolisieren die Kopplung der Ein- und Ausgangsalphabete. Der Name des zu koppelnden Ereignisses wird an diese Verbindungslinien gesetzt, wobei kursive Schrift für den Namen im erzeugenden Modul und normale Schrift für den Namen im aufnehmenden Modul steht. Sind diese identisch, kann letzterer weggelassen werden (wie z.B. beim Signal-Tracking-Modul). Sich kreuzende Linien gelten als nicht verbunden. Abzweigungen werden mittels Punkten auf den Kreuzungen dargestellt.

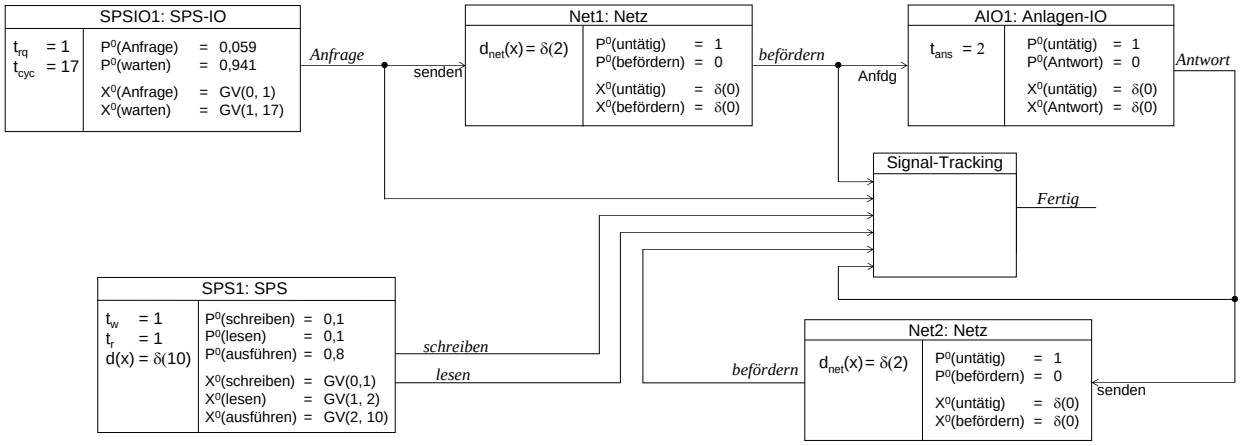

Abbildung 42: Instanziierung und Parametrierung für das Grundsystem

Die gesuchte Antwortzeitverteilung findet sich in Abbildung 43 mit ausgefüllten Quadraten markiert (rechter Graph). Die zugehörige Berechnung wurde auf einem PC mit Intel Pentium4 3 GHz Prozessor und 1 GByte Hauptspeicher durchgeführt und benötigte für Modellerstellung und 45 Datenpunkte insgesamt 12 Sekunden (Verifikation mittels bounded Until ohne absolute Zeitvariable). Die minimale Antwortzeit ergibt sich als Summe aus SPS-I/O-Zyklusdauer und einer Anlagen-I/O-Antwortzeit (vgl. hierzu auch das im rechten Teil der Abbildung 55 gezeigte GANTT-Chart). Deutlich zu erkennen sind außerdem die beiden, jeweils eine SPS-I/O-Zyklusdauer langen, Plateaus. Dass die Werte innerhalb der beiden Plateaus gleichbleiben, hat zweierlei Ursache: Zum einen überdeckt der SPS-I/O-Zyklus den SPS-Zyklus und zum anderen tritt die SPS-I/O-Verzögerung nur beim ersten Sendevorgang als Gleichverteilung über ihrer Zykluslänge auf, während sie beim zweiten Mal als Funktion der bis dahin vergangenen Zeit auftritt. Werte größer als 53 ms treten nicht auf.

Zum Vergleich wurden die Ergebnisse des, bereits in Abschnitt 5.2 vorgestellten, direkt verdrahteten Systems ebenfalls dargestellt (linker, mit Rauten markierter Graph). Da dieses System lediglich aus einer SPS, einem Sensor und einem Stellglied bestand, erklären sich Plateaubreite und Minimalzeit direkt aus der Zyklusdauer der SPS.

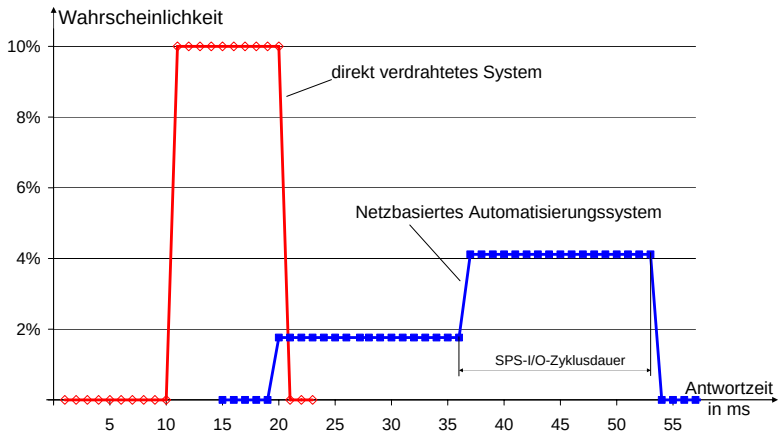

Abbildung 43: Antwortzeitverteilung: Diskrete Wahrscheinlichkeit dafür, dass die Antwortzeit des Systems im Bereich eines bestimmten Zeitintervalls der Länge einer Millisekunde liegt; für das NAS-Grundsystem (rechter, mit ausgefüllten Quadraten markierter Graph) und das, in Abschnitt 5.2 vorgestellte, direkt verdrahtete System (linker, mit Rauten markierter Graph).

6.1.3 Relative Anteile der Verhaltenseigenschaften

In diesem Abschnitt soll untersucht werden, welchen relativen Anteil Bearbeitung und Netzpassagen an der Antwortzeit eines NAS haben. Als **Bearbeitung**szeiten gelten die Reaktionszeit der Anlagen-I/O (2 ms) sowie die Zykluszeiten der beteiligten SPSen. Hierbei wird angenommen, dass die SPS in den untersuchten Szenarien die Reaktion auf einen geänderten Eingangswert innerhalb eines Zyklus berechnet (z.B. O1 := I1). Die **Netz**zeit entspricht dem Mittelwert von d_{net} (Abbildung 38) multipliziert mit der Anzahl der Netzpassagen und beträgt für das Grundsystem 4 ms. Die verbleibende Zeit wird als **Synchronisation**sverzögerungen klassifiziert. Diese werden durch die Überlagerung nicht synchronisierter Arbeitszyklen verursacht und ergeben sich als Differenz der berechneten mittleren Antwortzeit und der Summe aus Netz- und Bearbeitungszeit.

Hinweis: Eine ähnliche Aufteilung findet sich bei [Marsal, 2007], welche jedoch nicht mit Mittelwerten arbeitet, sondern mit dem jeweils simulativ bestimmten best- und worst-case.

Tabelle 6: Aufteilung der Antwortzeiten für herkömmliche und NAS-basierte SPS

	direkt verdrahtete I/Os		I/Os über das Netzwerk	
Netz	–	–	4,0 ms	10,0%
Bearbeitung	10,5 ms	67,74%	14,5 ms	36,3%
Synchronisation	5 ms	32,26%	21,4 ms	53,7%
Summe / Zuwachs	15,5 ms	–	39,9 ms	+166,7%

Die Aufteilung der mittleren Antwortzeit nach Ursachen ist in Tabelle 6 jeweils als absoluter sowie prozentualer Anteil (bezogen auf die Summe) dargestellt. Zum Vergleich wurden noch die Zeiten und Anteile des direkt verdrahteten Systems aufgeführt. Selbstverständlich weist auch dieses System bereits durch das zyklische Eigenverhalten der SPS verursachte Synchronisationsverzögerungen auf, allerdings sind diese deutlich geringer als im Falle eines NAS. Deutlich wird damit, dass alle Vorteile der neuen Automatisierungsstruktur mit einer Erhöhung der Antwortzeit bezahlt werden müssen. Auffällig ist hierbei, dass diese Erhöhung deutlich größer ist als der durch die Netzpassagen verursachte Zeitverlust von 4 ms. Die Zeit, die durch Warten eines Teilprozesses auf die Synchronisation mit einem anderen in NAS verbracht wird, liegt sogar deutlich über der Bearbeitungszeit.

6.2 Stochastische Übergangszeiten

Von den vier, in Kapitel 1 diskutierten Formen stochastischen Erscheinens, wurde für die bisherigen Modelle lediglich die stochastische Verteilung des Anfangszustands verwendet. In diesem Abschnitt wird nun eine weitere Erscheinungsform hinzugefügt, nämlich die Verhaltensabstraktion durch Verzögerungsverteilungsdichten. Beispielhaft werden eine variierende Netzübertragungsdauer und ein flexibler SPS-Zyklus besprochen.

6.2.1 Variable Netzlaufzeit

Für die Netzübertragung wird eine UDP-Übertragung angenommen, welche – wie Messungen von [Gottheit, 2006] für LAN und [Bohl, 2006] für WLAN gezeigt haben – die in Abbildung 44 dargestellten relativen Verzögerungshäufigkeiten aufweisen.

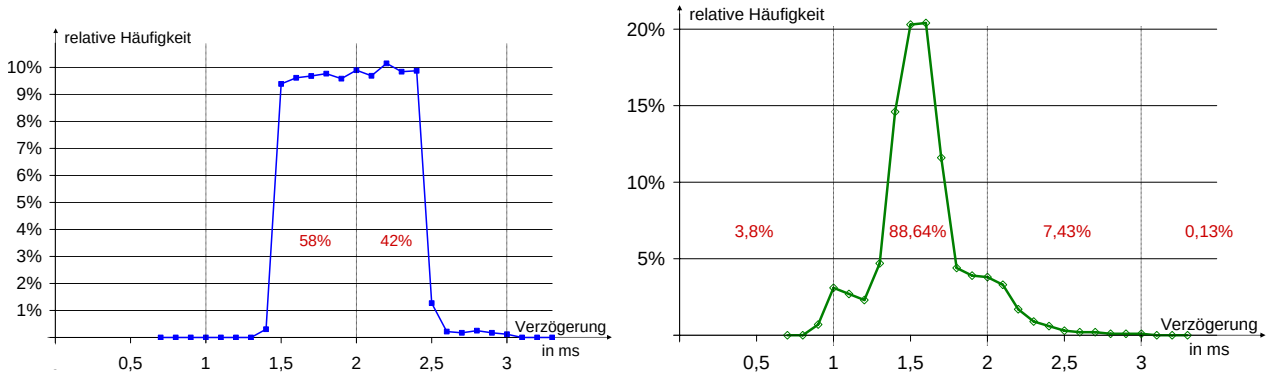

Abbildung 44: Gemessene Verzögerungszeiten von UDP-Übertragungen für LAN (links, [Gottheit, 2006]) und WLAN (rechts, [Bohl, 2006]). Angegeben sind außerdem die, bei einer Diskretisierung mit $\Delta t = 1$ ms entstehenden, prozentualen Verzögerungsanteile.

Für Messungen anderer Netzwerke wird auf die Literatur verwiesen. Beispielhaft seien genannt: [Maciej, 2006] für CAN, [Artell et al., 2005] für GPRS und [Antolović et al., 2006] für PROFINET.

Während sich im (zeitkontinuierlichen) DesLaNAS-Modell nur die Parametrierung ändert, entsteht durch die veränderte Übertragungscharakteristik ein anderes diskretes Modell (Abbildung 45). Abbildung 47 illustriert den Unterschied der Antwortzeitverteilungen des Grundsystems bei Verwendung von konstanter versus variabler Netzlaufzeit.

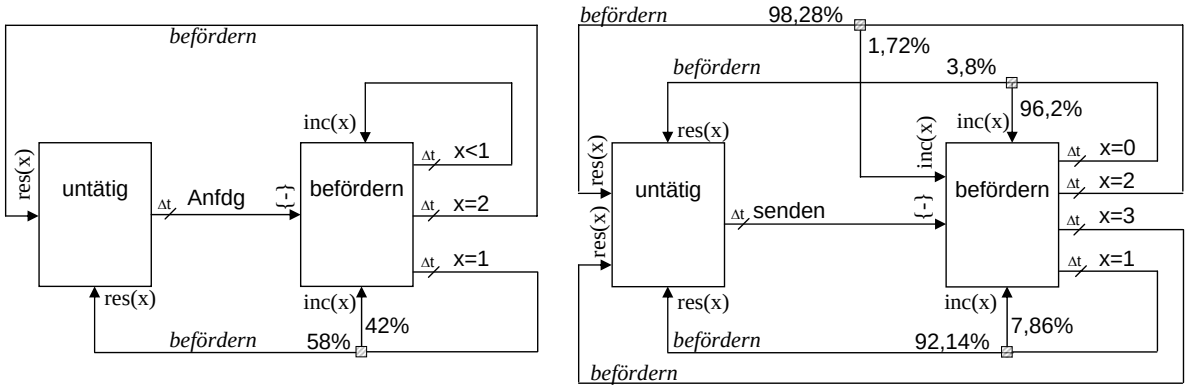

Abbildung 45: Netzmodul in zeitdiskreter Ausführung. Die Diskretisierung erfolgte mit einer Diskretisierungsschrittweite von $\Delta t = 1$ ms für eine LAN-Verzögerung nach [Gottheit, 2006] (links) und einer WLAN-Verzögerung nach [Bohl, 2006] (rechts).

Die Umsetzung nach PRISM verläuft wie in den bisherigen Beispielen und erfordert zwei Zustände, vier Uhrwerte und eine Ausgangsvariable. Allerdings gibt es, wie Abbildung 46 zeigt, eine deutlich elegantere Variante, die nicht nur zu einem kleineren Zustandsraum führt, sondern sich sogar generisch formulieren lässt. Hierdurch wird erreicht, dass bei Verwendung unterschiedlicher Verteilungsfunktionen d_{net} und Schrittweiten Δt, lediglich die Modellparameter (`Net1Max, pNet11, ..., pNet1n`) geändert werden müssen, nicht aber der Code.

```
module Net1
  Net1Zst:  [0..Net1Max] init 0; // 0: untätig, 1,2,...,Net1Max: befördern
  [Dt]    Net1Zst=0 & !SendenNet1  ->  Net1Zst'=0 ;
  [Dt]    Net1Zst=0 & SendenNet1   ->  pNet11:(Net1Zst'=1) + pNet12:(Net1Zst'=2) + ...
                                         ... + pNet1n:(Net1Zst'=func(min,Net1Max,n)) ;
  [Dt]    Net1Zst>0                ->  Net1Zst'=Net1Zst-1 ;
  [post]  true                     ->  Net1Zst'=0 ;
endmodule
```

Abbildung 46: PRISM-Code für stochastische Netzübertragungszeit

In diesem Code ist `Net1Zst=0` mit dem Zustand `untätig` assoziiert und `Net1Zst=`$\{1,2,3,4,\dots\}$ mit dem Zustand `befördern`. `Net1Zst=1` steht hierbei für jenen Zählerstand, von dem aus das Ausgangsflag *befördern* aktiv gesetzt wird. `Net1Zst=`$\{2,3,4,\dots\}$ repräsentieren hingegen jene Situationen, in denen die Rückkehr in den Zustand `untätig` noch $\{2,3,4,\dots\}$ Zeitschritte entfernt ist.

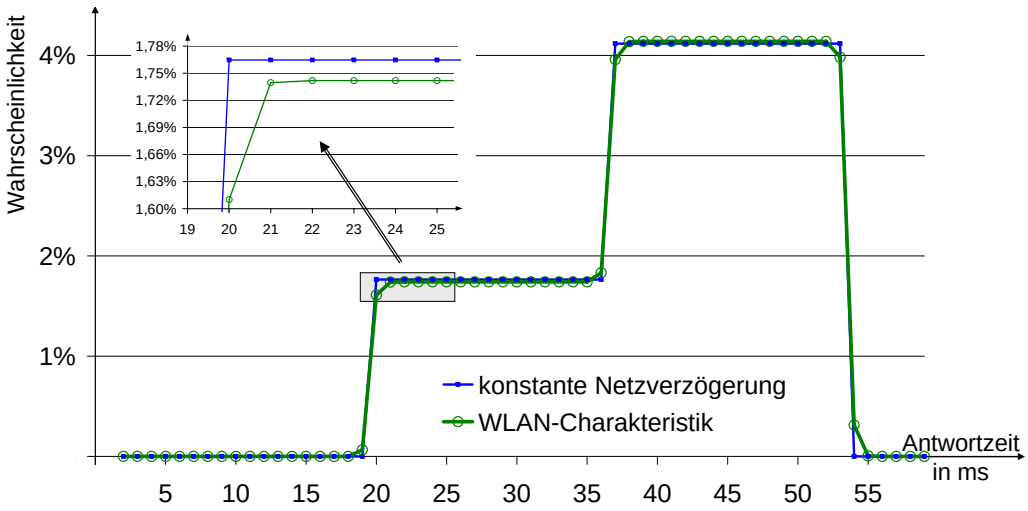

Abbildung 47: Vergleich der Antwortzeitverteilungen des Grundsystem unter Verwendung einer variablen Netzübertragungszeit mit WLAN-Charakteristik nach Abbildung 45 gegenüber einer konstanten Netzübertragungszeit von 2 ms.

6.2.2 Variabler SPS-Zyklus

Abbildung 48 zeigt die Antwortzeitverteilung für ein auf WLAN-basierendes Netz mit einer SPS, für deren Zyklus die folgenden Längen angenommen wurden: 9 und 11 ms mit je 20%-iger Wahrscheinlichkeit und 10 ms mit 60%. Da der Effekt des variablen SPS-Zyklus eher gering ist, wird im Weiteren – wie zuvor – von einem konstanten Zyklus ausgegangen und auch die Netzverzögerung wieder zu konstant 2 ms angenommen.

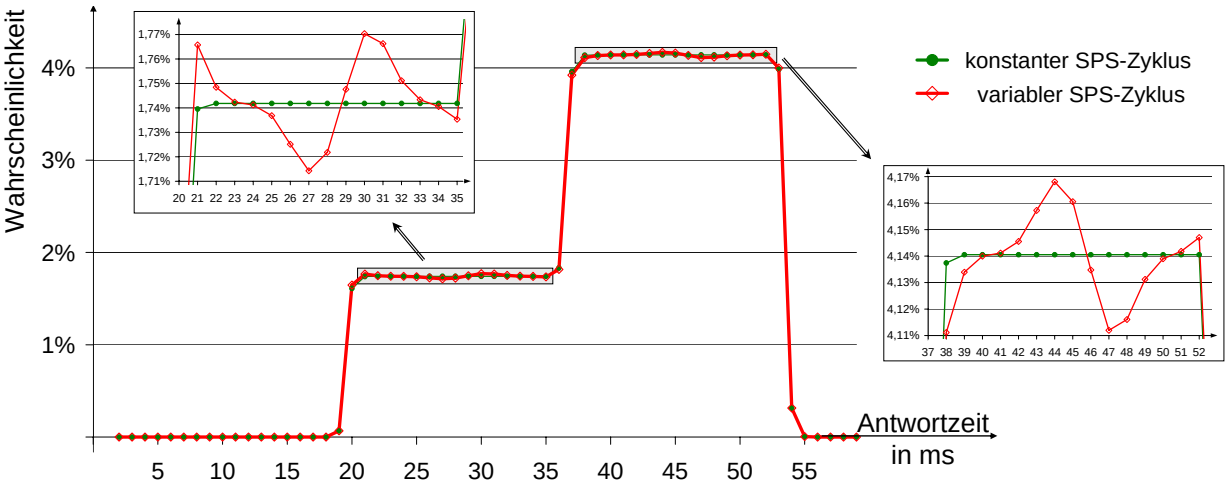

Abbildung 48: Vergleich der Antwortzeitverteilungen von variabler und konstanter SPS-Zyklusdauer, jeweils unter Verwendung der WLAN-Übertragungscharakteristik. Die beiden Ausschnitte zeigen eine Vergrößerung des Verlaufs an den Plateaus.

6.3 Stochastische und parallele Funktionsweisen

Wurde bisher lediglich das Grundsystem betrachtet, sollen nun auch Zugriffskonflikte und Fehler berücksichtigt werden. Zugriffskonflikte kommen zustande, weil in NAS mehrere Komponenten auf ein und dieselbe Ressource zugreifen. Im Beispiel tritt ein Ressourcenkonflikt an jenen Anlagen-I/Os auf, die von verschiedenen SPS-I/Os angefragt werden. Der zugehörige Zeitanteil wird als Differenz aus der für den ungestörten aber konfliktbehafteten Fall ermittelten mittleren Antwortzeit („Konfliktzeit“) und der mittleren Antwortzeit des ungestörten und zugriffskonfliktfreien Falls („Grundzeit“) ermittelt. Selbstverständlich unterscheidet sich diese Differenz von der tatsächlich auftretenden durchschnittlichen I/O-Wartezeit, da durch die erhöhte Wartezeit vor der Anlagen-I/O auch der Ankunftszeitpunkt an der SPS und folglich derjenige an der SPS-I/O verschoben wird (in den analysierten Beispielen war diese Größe stets etwas kleiner). Da die hierbei zusätzlich auftretenden Verzögerungszeiten jedoch immer durch „Warten auf eine Ressource“ hervorgerufen werden, ist die Unterscheidung zwischen Zugriffskonflikt und Synchronisation primär didaktischer Natur, um den Effekt einer Mehrbenutzerumgebung demonstrieren zu können.

Ähnliches gilt für durch Fehler hervorgerufene Verzögerungen. Diese werden insofern analog zu Zugriffskonflikten behandelt, als dass die durch Fehlfunktion im Mittel zusätzlich zur Grundzeit auftretende Antwortzeit als Fehlerverzögerung benannt wird. Der verbliebene Fall einer Überlagerung von Fehlern und Zugriffskonflikten wird zunächst nach Zugriffskonflikt und dann nach Fehlern aufgegliedert und ist somit lediglich anschaulicher Natur. Aufgrund der Diskretisierung kommen mit den Zugriffskonflikten auch stochastische Entscheidungsprozesse zwischen zwei „gleichzeitig“ eintreffenden Signalen zur Analyse. Und schließlich findet sich in der Analyse von Fehlern auch der letzte der vier in Kapitel 1 eingeführten, stochastische Beschreibungsformen erfordernden, Aspekte, nämlich derjenige des unvorhersagbaren Verhaltens.

6.3.1 Fehler

Die in einem NAS auftretenden Ausfälle und Fehler können sowohl unter dem Gesichtspunkt des Auftretens, als auch der Dauer in zufällig und konstant aufgeteilt werden. Während bei konstantem Auftreten der Auftretenszeitpunkt als bekannt angesehen werden kann, lassen sich im stochastischen Fall die Eintrittszeitpunkte lediglich mittels (auf Erfahrung oder Messung basierenden) Wahrscheinlichkeitsverteilungen angeben. In analoger Weise verhält es sich mit der Dauer.

Stellvertretend für die Vielzahl möglicher Fehler seien dem Grundsystem zwei Fehlerquellen hinzugefügt: Zum einen kann mit einer (diskreten) Wahrscheinlichkeit von $p_{IOF} = 10^{-3}$ ein Sensorwert ungültig sein (Anlagen-I/O-Fehler). Dies bedeutet, dass dieser Wert erst bei der nächsten Anfrage durch die SPS-I/O korrekt gelesen werden kann. Zum zweiten kann die Netzübertragung mit einer mean time between failures (MTBF) von 10 s für eine Dauer von $F_t = 4$ ms ausfallen (Netzfehler), wobei ein sofortiger Wiederausfall mit $p_w = 10^{-4}$ möglich ist. Alle zu übertragenden Pakete werden aber übertragen, sobald das Netzwerk wieder zur

Verfügung steht. Die zugehörige Ausfallcharakteristik wird durch die Exponentialfunktion $\lambda_F \cdot e^{-\lambda_F \cdot x}$ beschrieben, deren Parameter λ_F sich aus der MTBF von 10 s bestimmen lässt.

Während sich die Anfangsverteilung des I/O-Fehlerfalles direkt aus der Ausfallwahrscheinlichkeit ermitteln lässt (aktuelle Ausfallrate ist stets $p_{IOF} = 10^{-3}$), muss diese im Netzfehlerfall erst analytisch bestimmt werden. Dies ist jedoch relativ einfach, da der Ausfallautomat repetitiv ist und somit der stationäre Fall gilt. Zuverlässigkeitsdaten wie sie hier Verwendung finden, können nach [DIN EN IEC 61508-6, Teil B3] ermittelt werden, welcher die Wahrscheinlichkeit eines Ausfalls je Stunde (für die Betriebsart mit hoher Anforderungsrate oder kontinuierlicher Anforderung) beschreibt.

Der Fehlertypus des Netzfehlers lässt sich als separates Modul modellieren, auf welches dann sämtliche Netzmodule synchronisieren (vgl. Abbildung 49, links). Dies macht insbesondere dann Sinn, wenn der auftretende Fehler mehrere Module beeinflusst, wie das hier für die beiden Netzübertragungen der Fall ist. Ein solches separates Fehlermodul hat zwei Zustände, nämlich `OK` und `Fehler`, wobei die Übergangswahrscheinlichkeiten durch entsprechende Dichtefunktionen gegeben sind.

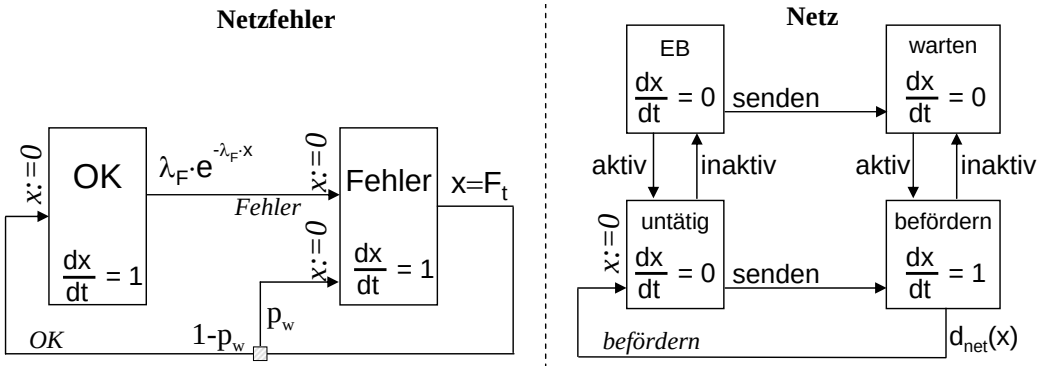

Abbildung 49: Netzfehlermodul (links) und einen Netzfehler berücksichtigendes Netzmodul (rechts)

Um den Fehlerfall repräsentieren zu können, muss das Netzmodul um zwei weitere Zustände ergänzt werden (Abbildung 49, rechts), wobei sich der Automat bei normalem Betrieb in einem der beiden ursprünglichen Zustände befindet. Ereignet sich ein Fehler (Ereignis: inaktiv), so wechselt der Automat vom aktuellen (unteren) in den direkt oberhalb des aktuellen befindlichen Zustand. Sobald der Fehler behoben wurde (Ereignis: aktiv) wechselt der Automat in den jeweils darunter liegenden Zustand. Selbstverständlich ist es möglich, dass eine Anfrage (senden) eingeht, während sich der Automat im Zustand `EB` (empfangs- aber nicht beförderungsfähig) befindet. In diesem Fall wechselt der Automat in den Zustand `warten`. Die umgekehrte Richtung ist nicht möglich, da ein Komplettverlust des Datenpakets nicht vorgesehen ist. In Bezug auf die Initialisierung dieser Module ist anzumerken, dass die fehlerbehafteten Netzmodule nun natürlich nicht mehr unabhängig initialisiert werden können, da ihr Initialzustand von dem des Netzfehlermoduls abhängt.

Der für die Anlagen-I/O angenommene Fehlertyp ist von lediglich lokaler Wirkung, d.h. er beeinflusst nur ein einziges Modul. Dafür führt er jedoch zum Komplettverlust der Infor-

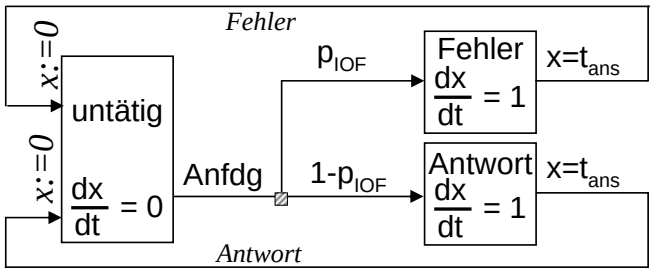

Abbildung 50: Automatendarstellung einer Fehler berücksichtigenden Anlagen-I/O

mation; ohne dass hierdurch das übergeordnete Zeitverhalten des Moduls verändert würde. Dieses Verhalten wird durch den in Abbildung 50 gezeigten Automaten repräsentiert.

Während sich die Verwendung eines nicht zum Informationsverlust führenden Fehlertyps (z.B. Netzfehler) nicht im Signal-Tracking-Modul niederschlägt, muss dieses für einen Information vernichtenden Fehlertyp, wie in Abbildung 51 gezeigt, erweitert werden, da ein Zurückkehren zu einem bereits passierten Zustand erforderlich wird. Schließlich müssen die Module noch instanziiert, verbunden und parametriert werden (Abbildung 52).

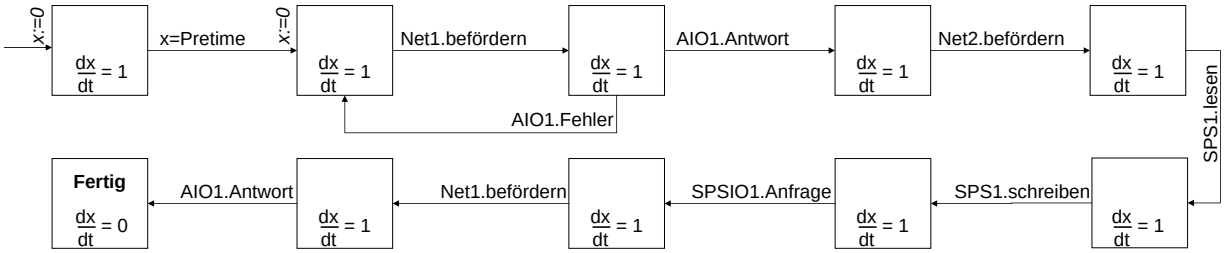

Abbildung 51: Signal-Tracking unter Berücksichtigung eines Anlagen-I/O-Fehlers

Tabelle 7 zeigt, welche zusätzliche Verzögerung sich im (nicht zugriffskonfliktgestörten) System durch die unterschiedlichen Fehlerarten einstellt. Als dritte Spalte („Anteil über Max“) wurde hierbei noch der Anteil der Fälle aufgetragen, in denen die Antwortzeit größer als die maximale Antwortzeit des nicht fehlerbehafteten Vergleichssystems ist. Die zugehörigen Antwortzeitdistributionen sind in Abbildung 53 in halblogarithmischem Maßstab gezeigt. Obwohl die Antwortzeit durch einen Fehler teilweise erheblich vergrößert werden kann, muss dies natürlich stets in Relation mit der sehr niedrigen Auftretenswahrscheinlichkeit gesehen werden.

Tabelle 7: Änderung der mittleren Antwortzeit unter Berücksichtigung von Fehlern

	absoluter Zuwachs	relativer Zuwachs	Anteil über Max
Sensor	17,0170 μs	0,0426%	0,07003%
Netzwerk	2,5295 μs	0,0063%	0,00412%

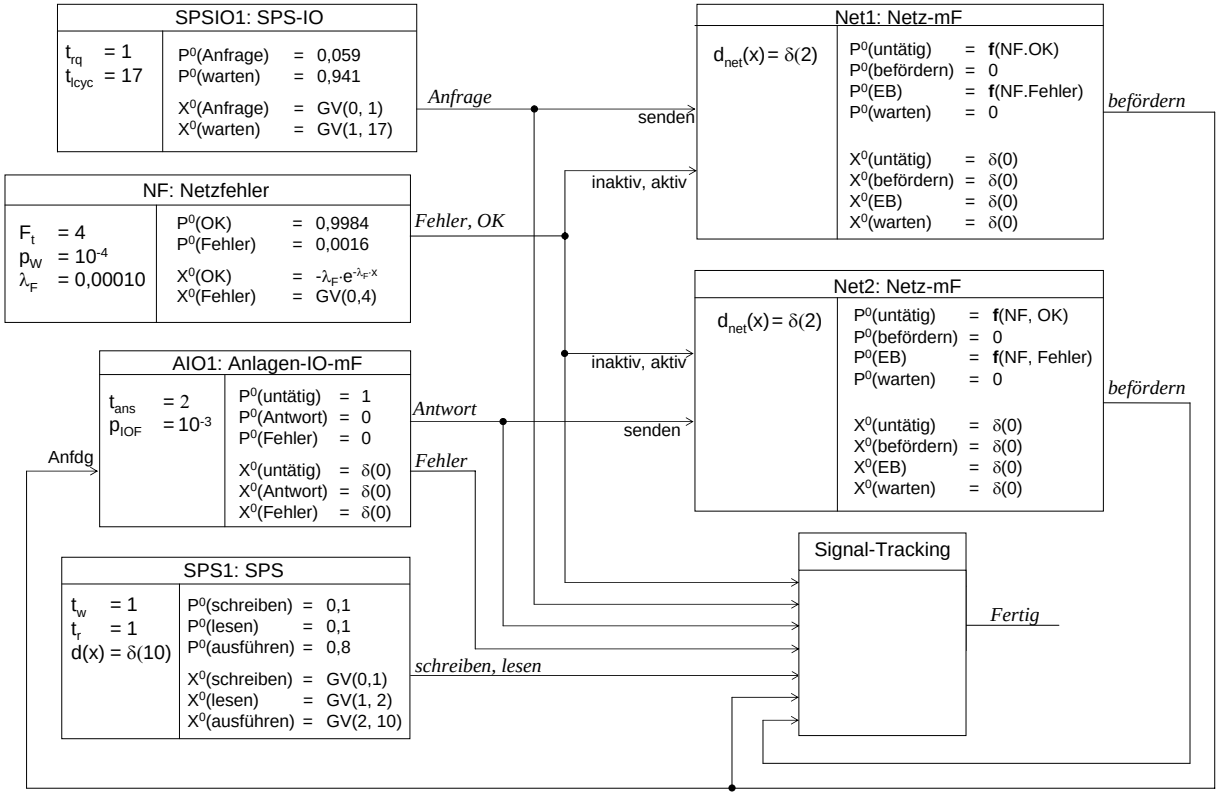

Abbildung 52: Instanziierung und Parametrierung des NAS unter Berücksichtigung von Fehlern. f(NF.OK) zeigt an, dass dieser Anfangszustand vom Zustand `OK` des Moduls NF abhängt.

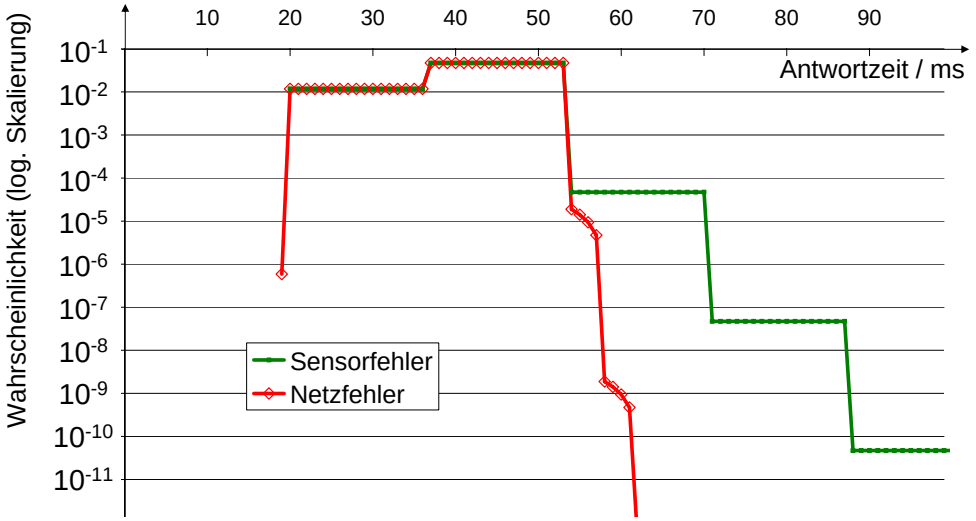

Abbildung 53: Antwortzeitverhalten des fehlerbehafteten aber zugriffskonfliktfreien Systems

6.3.2 Zugriffskonflikte

Die bisher betrachtete Situation eines alleinigen (oder zumindest priorisierten) Zugriffs der SPS auf die anzufragende Anlagen-I/O taucht in der Praxis selten auf. Stattdessen finden sich mehrere SPSen, die unabhängig voneinander diese Anlagen-I/O anfragen. Dies ist auch insbesondere unter dem Erweiterungsaspekt interessant, da es in NAS möglich ist, dynamisch unterschiedlich viele Teilnehmer vorzufinden. PMC ist dabei eine Möglichkeit, die Konsequenzen derartiger Veränderungen a-priori beurteilen zu können.

Ist die Anlagen-I/O beim Eintreffen einer Anfrage gerade mit der Bearbeitung einer anderen Anfrage beschäftigt, so wird zunächst diese, sich bereits in Bearbeitung befindliche Anfrage abgearbeitet und erst anschließend alle später eingetroffenen Anfragen (in der Reihenfolge ihres Eintreffens). Zur Abbildung dieses Verhaltens muss das DesLaNAS-Modul der Anlagen-I/O um einen Anfrage-Puffer erweitert werden. Die Struktur des Signal-Tracking Moduls ändert sich für die Berücksichtigung von Zugriffskonflikten nicht. Die Instanziierung der neu hinzukommenden SPSIO2 und SPSIO3 erfolgt wie für die bereits vorhandene SPSIO1, mit dem Unterschied, dass für die zweite SPS-I/O eine Zykluszeit von 19 ms und für die dritte von 17 ms angenommen wurde. Darüber hinaus werden noch je ein weiteres Netzmodul benötigt, um den Datentransport von der zweiten bzw. dritten SPS-I/O zur Anlagen-I/O zu beschreiben. Der zugehörige Rücktransport spielt indes für den betrachteten Ablauf keine Rolle und wird daher nicht modelliert.

Abbildung 54 illustriert die Antwortzeitverteilungen für eine und drei SPSen. Ein Vergleich derartiger Häufigkeitsverteilungen zeigt, in welchem Zeitbereich die zusätzlichen Verzögerungen in welchem Umfang auftreten. Eine genaue Analyse ergibt, dass die bei zwei bzw. drei SPSen entstehenden Zugriffskonflikte die mittlere Antwortzeit um 0,5% bzw. 1,1% erhöhen, während der prozentuale Anteil der Antworten die um mehr als 53 ms (Maximalverzögerung bei einer SPS) verzögert werden, 0,43% bzw. 0,67% beträgt (vgl. Tabelle 8).

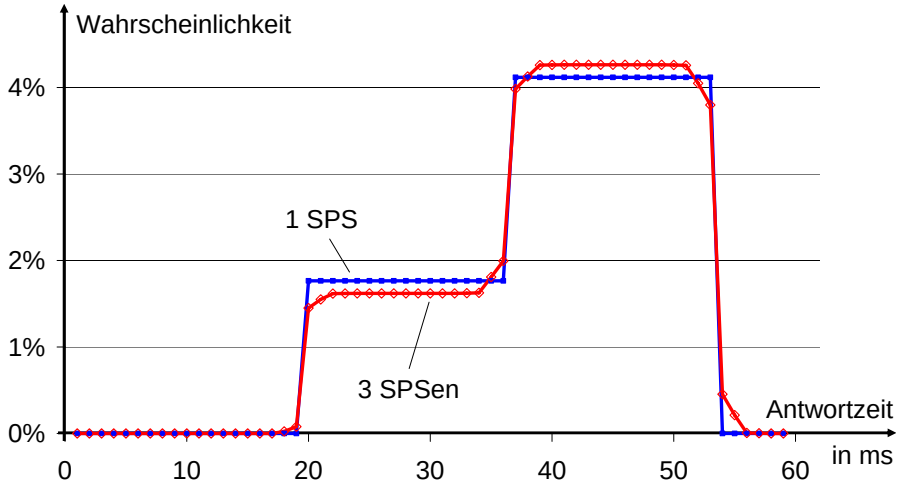

Abbildung 54: Antwortzeitverhalten des aus einer SPS bestehenden (zugriffskonfliktfreien) sowie des aus drei SPSen bestehenden (zugriffskonfliktbehafteten) Systems.

Tabelle 8: Aufteilung der mittleren Antwortzeiten unter Berücksichtigung von Zugriffskonflikten

	kein Konflikt		Zwei SPSen		Drei SPSen	
Netz	4 ms	10,0%	4 ms	9,97%	4 ms	9,92%
Bearbeitung	14,5 ms	36,3%	14,5 ms	36,16%	14,5 ms	35,94%
Synchronisation	21,4 ms	53,7%	21,4 ms	53,37%	21,4 ms	53,05%
Zugriffskonflikt	–	–	0,2 ms	0,50%	0,44 ms	1,09%
Summe / Zunahme	39,9 ms	–	40,1 ms	+0,5%	40,34 ms	+1,1%
Min.- / Max.-Wert	20 ms	53 ms	18 ms	55 ms	18 ms	57 ms
Anteil unter / über	–	–	0,9‰	4,3‰	1,0‰	6,7‰

Im Hinblick auf Sicherheitsaspekte sollte besonders beachtet werden, dass Zugriffskonflikte nicht nur die maximale Antwortzeit erhöhen, sondern auch die minimale Antwortzeit erniedrigen können. Wie Abbildung 55 verdeutlicht, liegt dies daran, dass die Anlagen-I/O beim Eintreffen einer Anfrage gerade mit der Bearbeitung der Anfrage einer anderen SPS-I/O beschäftigt sein kann. Dargestellt ist hierbei die zwischen den einzelnen (auf der Senkrechten aufgetragenen) Komponenten stattfindende Kommunikation in diskreten Zeitschritten. Beispielsweise sendet die SPS1-I/O zunächst ans Netz, welches für den Transport zwei Zeitschritte benötigt (vgl. Abbildung 55, rechts). Nach Ablauf dieser Zeitperiode wird die Anfrage an die Anlagen-I/O weitergereicht von wo aus sie nach erfolgter Bearbeitung zurück ans Netz und von dort an die SPS1 befördert wird.

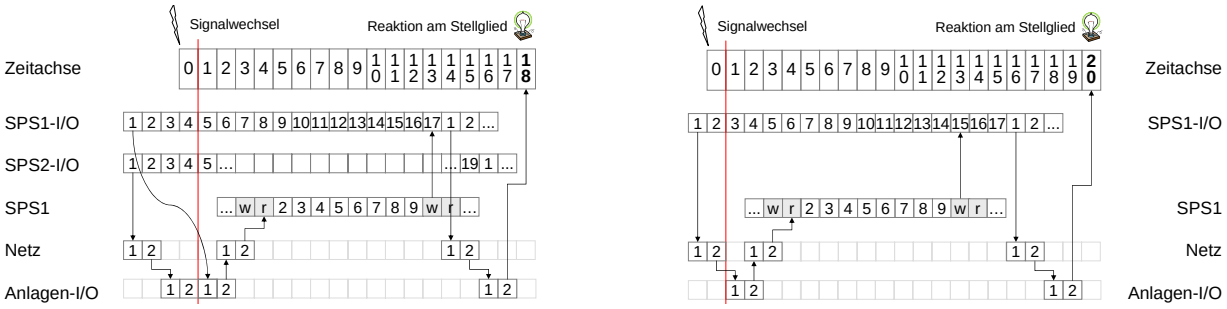

Abbildung 55: GANTT-Chart zur Entstehung der minimalen Antwortzeit im zugriffskonfliktbehafteten (links) und zugriffskonfliktfreien (rechts) Fall

Tritt hierbei der Fall ein, dass die Anlagen-I/O beim Eintreffen einer Anfrage der SPS1-I/O gerade mit der Bearbeitung der Anfrage einer anderen SPS-I/O beschäftigt ist, so muss die betrachtete Anfrage warten, bis alle früher eingetroffenen Anfragen abgearbeitet worden sind. Erfolgt nun, wie im linken Teil der Abbildung 55 gezeigt, eine Änderung des Sensoreinganges während dieser Warteperiode, so wird diese Änderung noch vor der Bearbeitung

der gerade wartenden Anfrage registriert, wohingegen diese Anfrage im zugriffskonfliktfreien Fall noch den alten Wert zurückgemeldet hätte. In Bezug auf den Zeitpunkt der Sensorsignaländerung erhält die SPS also früher Bescheid, als dies im konfliktfreien System der Fall wäre. Ein solcher Effekt tritt zwar selten ein (im betrachteten Beispiel nur mit einer Wahrscheinlichkeit von ca. 0,1%), senkt aber die minimale Antwortzeit von 20 auf 18 ms (vgl. Tabelle 8). Abbildung 55 zeigt jedoch auch, dass bei der vorliegenden Parameterwahl keine Antwortzeit unter 18 ms möglich ist, d.h. dass sich selbst bei drei oder vier konkurrierenden SPSen keine Änderung der minimalen Antwortzeit einstellt; wohl aber der Wahrscheinlichkeit, dass eine Antwortzeit von 18 ms auftritt.

Anders verhält sich dies bei der Maximalzeit, wie Abbildung 56 beispielhaft für den Fall eines Zugriffskonflikts durch drei SPSen zeigt. Wie man der Grafik entnehmen kann, hat nur der dritte Zugriff der SPS1-I/O auf die Anlagen-I/O-Karte (rechts unten dargestellt) einen Einfluss auf die maximale Antwortzeit. Jeder mögliche Zugriffskonflikt verlängert die Antwortzeit also um zwei Millisekunden von 53 ms im konfliktfreien System auf 55 ms bei 2 SPSen, auf 57 ms bei 3 SPSen usw. Dass die beiden anderen Zugriffe auf die Anlagen-I/O keinen Einfluss auf die Maximalzeit haben, liegt daran, dass der aus SPS- und SPS-I/O-Zykluszeiten entstehende Puffer groß genug ist, um die durch Zugriffskonflikte auftretende Zeitverschiebung aufzunehmen. Der Abbildung 56 ist diesbezüglich zu entnehmen, dass das Ergebnis von der SPS spätestens im 48. Takt der Zeitachse ausgeschrieben werden muss, um noch beim nächsten SPS-I/O-Sendezeitpunkt berücksichtigt werden zu können. Da das Schreiben im konfliktfreien System im 38. Takt erfolgen würde, beträgt der Puffer für die gewählte Konfiguration also 10 ms. Wird dieser jedoch überschritten, verlängert sich die maximale Antwortzeit umgehend um einen kompletten SPS-I/O-Zyklus (vgl. hierzu auch [Greifeneder und Frey, 2007a]).

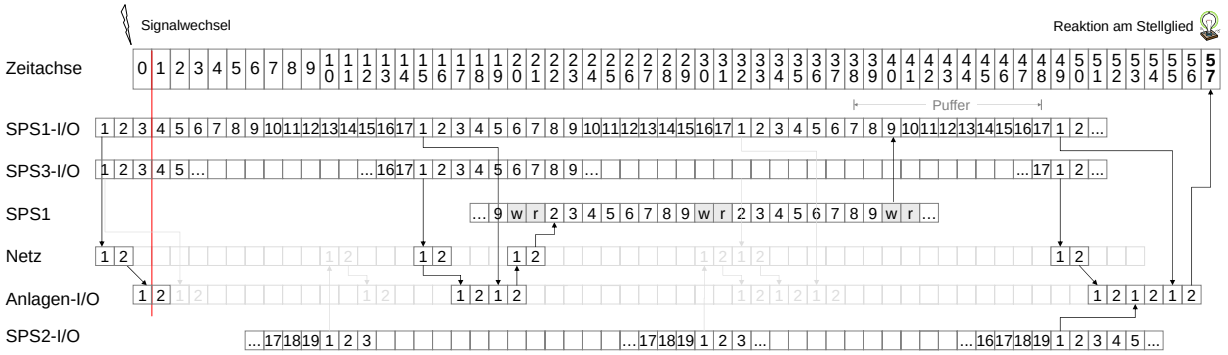

Abbildung 56: GANTT-Chart zur Entstehung der maximalen Antwortzeit im zugriffskonfliktbehafteten Fall

Abbildung 57 zeigt zusammenfassend die Aufteilung der mittleren Antwortzeit auf die eingeführten Kategorien. Im oberen Diagramm ist hierbei ein zugriffskonflikt- und fehlerfreies System dargestellt, im unteren Diagramm hingegen der Zugriff von drei SPSen samt sich überlagerndem Sensor- und Netzfehler. Deutlich zu erkennen ist, dass der mittlere Einfluss von Fehlern und Zugriffskonflikten eher gering ist und dass das Netzwerk bei weitem nicht jenen Einfluss hat, der ihm oftmals zugerechnet wird. Klar ist aber auch, dass ohne das

Netzwerk die gezeigten Strukturen und Effekte nicht auftreten würden und Fehler ebenso wie Zugriffskonflikte im Einzelfall zu durchaus beachtlichen Verzögerungen führen können.

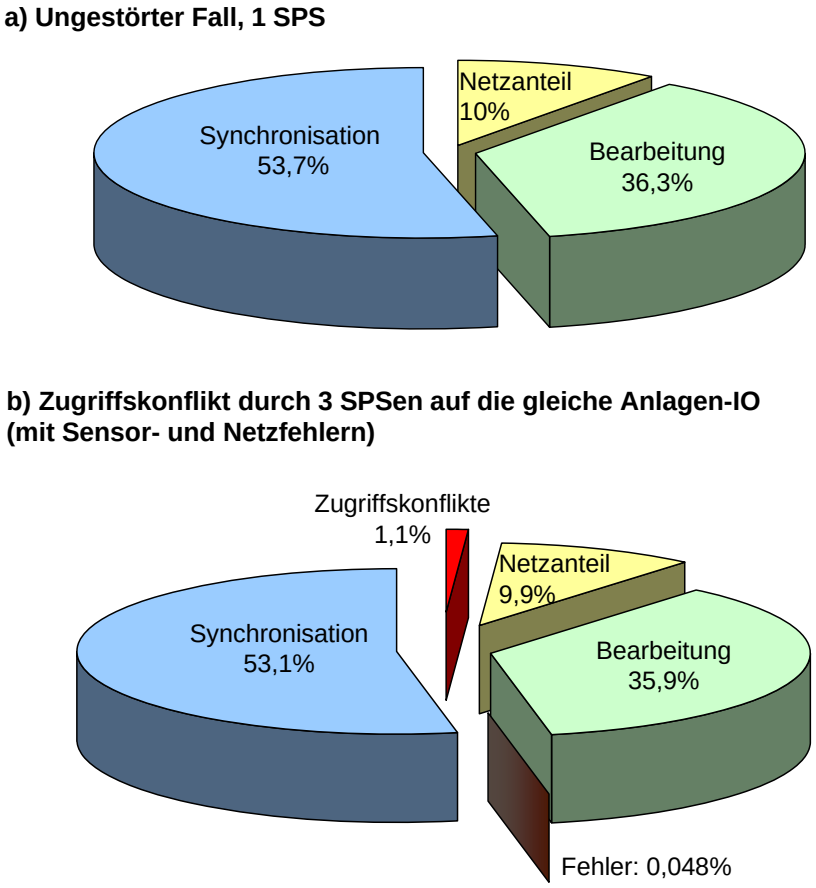

Abbildung 57: Prozentuale Aufgliederung der mittleren Antwortzeit nach Einflussgrößen

6.4 Weiterführende Anwendungsmöglichkeiten

Zweifelsohne eröffnet die Anwendung einer stochastischen und zeitbewerteten Analysemethodik eine ganze Reihe von faszinierenden Anwendungsmöglichkeiten. Hierzu gehören beispielsweise Fragen der Konzeption neuer Anlagen, der Re-Konfiguration (z.B. bei Erweiterung der Anlage), sowie Methoden der Anlagenoptimierung unter Qualitätsaspekten (vgl. hierzu auch [Greifeneder und Frey, 2006b]). Selbstverständlich kann an dieser Stelle nur ein winziger Bruchteil der Möglichkeiten dargestellt werden. Exemplarisch wurden zwei Fragestellungen der Konzeptionsphase ausgewählt, namentlich eine Konfigurations- und eine Komponentenanforderungsanalyse, welche in den Abschnitten 6.4.1 und 6.4.2 diskutiert werden. Für darüber hinaus gehende Anwendungsfälle wird auf frühere Publikationen verwiesen. So findet sich beispielsweise in [Greifeneder und Frey, 2006a] die Antwortzeitanalyse eines aus zwei Switchen und acht Anlagen-I/Os bestehenden zugriffskonfliktbehafteten

Netzbasierten Automatisierungssystems. In [Greifeneder und Frey, 2007b] wird hingegen u.a. die Frage diskutiert, welchen Einfluss unterschiedlich lange Zykluszeiten auf die Antwortzeitverteilung haben. In [Greifeneder und Frey, 2007c] wird schließlich jener Spezialfall diskutiert, bei dem das Stellglied direkt an die SPS angeschlossen ist, während der Sensor weiterhin über das Netzwerk angesprochen wird. Der Vergleich einer solchen Konstellation mit Messwerten findet sich in [Greifeneder und Frey, 2006d].

6.4.1 Beispiel zur Konfigurationsanalyse

In NAS ist es möglich, die Abarbeitung von Steuerungsalgorithmen auf mehrere SPSen zu verteilen. Abhängig von der Programmstruktur und den internen Abhängigkeiten ergeben sich hier vielfältige Möglichkeiten der Optimierung. Ein kleines Beispiel soll zeigen, dass der Effekt auf die Antwortzeit nicht trivial vorhersagbar ist. Angenommen ein SPS-Programm besteht aus drei Programmteilen A, B und C mit entsprechenden Ausführungszeiten von acht, acht und einer Millisekunde. C benötige die Ergebnisse von A und B, A und B seien aber voneinander unabhängig (parallelisierbar). Mögliche Realisierungen wären z.B.:

1. Abarbeitung der Sequenz A,B,C auf einer SPS mit einer Zykluszeit von 19 ms (wobei davon ausgegangen wird, dass die Behandlung der Ein- und Ausgänge jeweils 1 ms in Anspruch nimmt) oder
2. Abarbeitung von A auf einer SPS mit einer Zykluszeit von 10 ms, die dann ihr Ergebnis an eine zweite SPS (Zykluszeit 11 ms) sendet, auf der die Sequenz B,C abgearbeitet wird.
3. Nebenläufige Ausführung von A und B auf zwei SPSen mit je einer Zykluszeit von 10 ms sowie Abarbeitung von C auf einer dritten SPS mit einem sehr schnellen Zyklus von 3 ms.

Die SPS-I/O-Zyklen wurden zu 17, 17 und 5 Millisekunden festgelegt. Darüber hinaus wurde vereinfachend angenommen, dass die SPS-I/O-Karten auch an sie gesandte Netzpakete annehmen können, ohne dass dies ihren Ausführungszyklus beeinträchtigen würde. Unter Verwendung dieser Annahme kann eine SPS-I/O das von der zugehörigen SPS ermittelte Teilergebnis im Rahmen ihrer normalen Anlagen-I/O-Telegramme direkt über das Netzwerk an die in der Bearbeitungsreihenfolge nächste SPS schicken.

Wie die in Abbildung 58 dargestellten Distributionen zeigen, ist die erste Variante sowohl in Bezug auf Mittel- als auch Maximalwert schneller als die zweite, hat allerdings zwei Nachteile, nämlich, dass die Zykluszeit einer SPS nicht beliebig veränderbar ist und dass im Befehlscode durchaus Anweisungen enthalten sein können, die die schnellere Abarbeitung benötigen. Am Besten schneidet die drei SPSen einsetzende Variante ab. Die zugehörige Verteilungsanalyse ist in Tabelle 9 dargestellt. Ein Vergleich der Ergebnisse ermöglicht hierbei die in Bezug zur Antwortzeit stehende Bewertung einer Konfigurationsänderung ausgehend von einer einzigen SPS, die alle Schritte abarbeitet, hin zu einer flexibleren und meist auch mächtigeren Struktur.

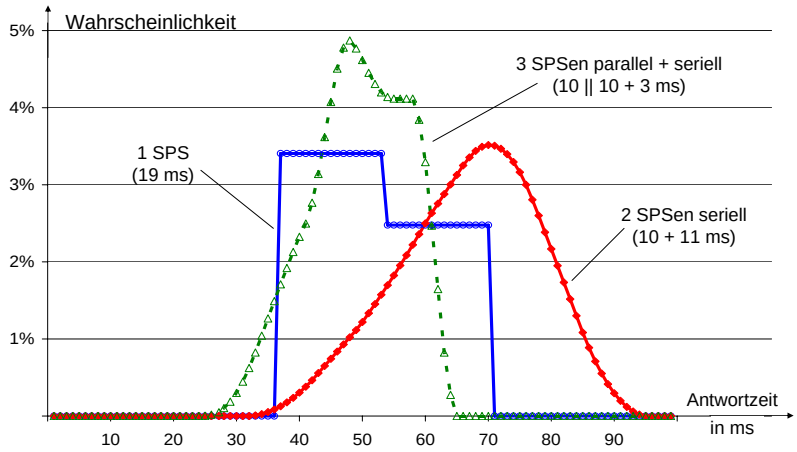

Abbildung 58: Antwortzeiten für die Verteilung von drei Programmteilen mit insgesamt 17 ms Bearbeitungszeit auf unterschiedlich viele SPSen

Tabelle 9: Aufteilung der mittleren Antwortzeiten für unterschiedliche Konfigurationen

	Netz-anteil	Bearbei-tung	Synchro-nisation	Werte-bereich	Mittelwert Streuung
1 SPS 19 ms	4 ms 7,7%	23,5 ms 45,0%	24,66 ms 47,3%	37 – 70 ms	52,16 ms 9,72 ms
2 SPSen 10 + 11 ms	6 ms 9,0%	25,5 ms 38,1%	35,4 ms 52,9%	33 – 94 ms	66,9 ms 11,2 ms
3 SPSen 10||10 + 3 ms	6 ms 12,2%	17,5 ms 35,5%	25,77 ms 52,3%	25 – 64 ms	49,27 ms 7,65 ms

Implementierung

Für eine auf zwei sequentiellen SPSen verteilte Berechnung müssen sowohl das System als auch das Signal-Tracking erweitert werden. Systemseitig sind (im Vergleich zur Grundkonfiguration) sowohl eine weitere SPS, eine weitere SPS-I/O als auch zwei weitere Netzmodule notwendig, die im Folgenden mit SPS2 und SPSIO2 bezeichnet werden. Im Signal-Tracking Automaten muss die Passage über die zweite SPS eingefügt werden. Da sich an den ersten Automatenzuständen und -übergängen nichts ändert, zeigt Abbildung 59 lediglich den erweiterten Teil des Signal-Tracking Automaten.

Die Modellierung einer auf zwei parallelisierten und einer sequentiellen SPS verteilten Berechnung ist im Vergleich deutlich aufwändiger. Um die Parallelität abbilden zu können, wird zunächst für jede der beiden parallelisierten SPSen ein eigenes Signal-Tracking Modul erstellt, welches dem in Abbildung 41 dargestellten – unter Auslassung des vorletzten Zustandes (Bearbeitung durch die Anlagen-I/O) – entspricht. Diese beiden Signal-Tracking Module seien mit sig1 sowie sig2 benannt. Darüber hinaus wird ein drittes Signal-Tracking

Modul – wie in Abbildung 60 dargestellt – benötigt, welches zunächst wartet, bis die beiden parallelisierten SPSen die Information verarbeitet und an die dritte SPS weitergeleitet haben (Ereignisse sig1.Fertig bzw. sig2.Fertig). Anschließend folgt der bereits bekannte Ablauf.

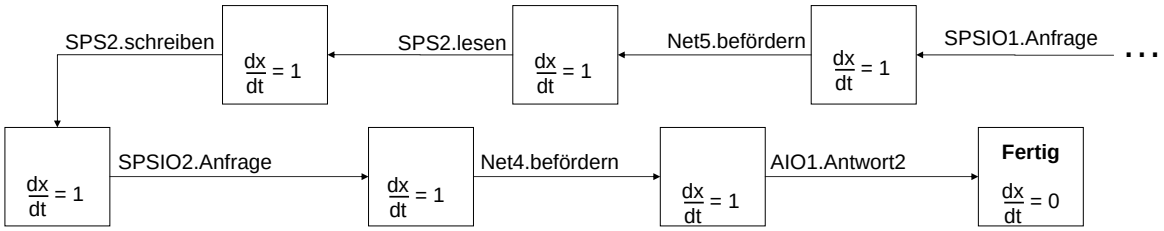

Abbildung 59: Signal-Tracking für sequentielle Bearbeitung auf zwei SPSen

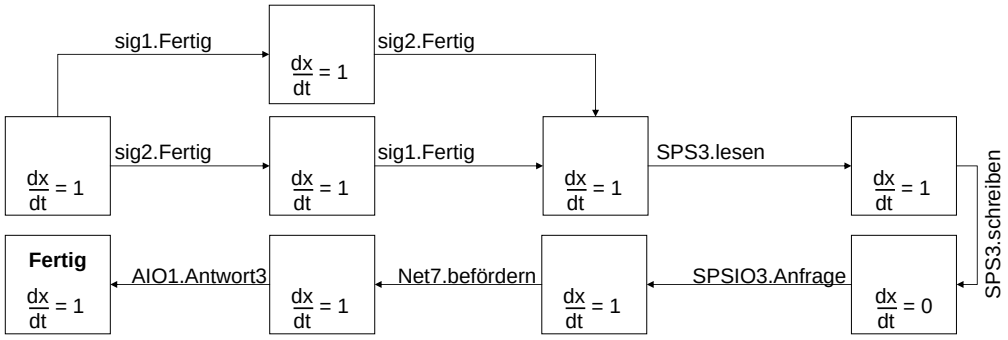

Abbildung 60: Überlagertes Signal-Tracking Modul für eine Bearbeitung auf zwei parallelisierten und einer dritten, sequentiell angeordnete SPS

6.4.2 Beispiel zur Komponentenanforderungsanalyse

Nimmt man für das an der Anlagen-I/O anzulegende Sensorsignal Typ II-Verhalten (fest vorgegebene Haltezeit) an, so kann der Sensorwert beim Eintreffen der Anfrage schon nicht mehr anliegen und somit nicht mehr ausgelesen werden, falls die Anlage nicht entsprechend schnell ausgelegt wurde. In diesen Fällen ist es interessant zu wissen, mit welcher Wahrscheinlichkeit, ein (mindestens) n Millisekunden lang gehaltenes Signal erkannt wird, bzw. wie lange ein Signal mindestens gehalten werden muss, damit es mit einer bestimmten Wahrscheinlichkeit erkannt wird. Hieraus lassen sich dann entweder Aussagen zum Verlustrisiko bei Verwendung einer gegebenen Konfiguration ableiten oder aber Anforderungen an das Sensorsignal bzw. die Hardware, formulieren.

Die Analyse dieses Problems ist verhältnismäßig einfach, da das Signal-Tracking auf die Elemente SPS-I/O, Anlagen-I/O und für den fehlerbehafteten Fall auf das Netzwerk, verkürzt wird. Tabelle 10 zeigt die Ergebnisse der zugehörigen Analyse für das bisherige Beispiel. In den Zeilen aufgetragen sind die Werte für verschieden viele anfragende SPSen. In den Spalten findet sich zunächst die minimale Haltezeit im ungestörten Fall, d.h., jene Zeit,

die das Signal mindestens anliegen muss, damit es im ungestörten Fall garantiert erkannt wird. Die folgenden vier Spalten geben hingegen jene Zeit an, die das Signal anliegen muss, um mit mindestens 99,999% bzw. 99,9999%-iger Wahrscheinlichkeit erkannt zu werden. Diese Werte entsprechen den über den Spalten angegebenen Verlustwahrscheinlichkeiten von 10^{-5} und 10^{-6}. Angegeben sind jeweils die Fälle eines Ausfalls der Anlagen-I/O (Sensorfehler) als auch eines Ausfalls des Netzwerkes. Erwartungsgemäß verringert sich die Verlustwahrscheinlichkeit mit steigender Haltezeit, bzw. erhöht sich die notwendige Haltezeit um unterhalb einer maximalen Verlustwahrscheinlichkeit zu bleiben.

Tabelle 10: Verlustwahrscheinlichkeiten in Abhängigkeit von Zugriffskonflikten und Fehlern

Anzahl SPSen	minimal erforderliche Haltezeit				
	ungestört	Fehler im Netzwerk		Anlagen-I/O-Fehler	
		$p_{\text{Verlust}} < 10^{-5}$	$p_{\text{Verlust}} < 10^{-6}$	$p_{\text{Verlust}} < 10^{-5}$	$p_{\text{Verlust}} < 10^{-6}$
1	20 ms	23 ms	24 ms	37 ms	37 ms
2	22 ms	23 ms	24 ms	37 ms	39 ms
3	24 ms	24 ms	25 ms	38 ms	40 ms

Eine ähnliche Diskussion ergibt sich für ein Sensorsignal mit Typ III-Verhalten, wobei dort zusätzlich die Signaländerungswahrscheinlichkeit berücksichtigt werden muss (vgl. hierzu [Greifeneder und Frey, 2005]).

6.5 Vergleich mit Messungen

Zur Validierung der Ergebnisse wurde das zugriffskonfliktfreie WLAN-System im Labor aufgebaut [Greifeneder et al., 2007], wobei die Zykluszeiten von SPS und SPS-I/O softwaremäßig vorgegeben wurden. Hierzu wurde das prinzipielle Verhalten von SPS und SPS-I/O jeweils auf einem eigenen Microcontroller (ATMEGA32) implementiert. In gleicher Weise sind die Anlagen-I/Os und der Signalgenerator, der auch die Verzögerungsbestimmung übernommen hat, auf eigenständigen Boards aufgebaut worden. Abbildung 61 zeigt die Struktur des Messaufbaus. Die Kommunikation wurde mittels WLAN realisiert, wobei als Sendebausteine WiPorts der Firma Lantronix Verwendung fanden. Angeschlossen wurden die WiPorts an die ATMEGA32 über eine serielle Schnittstelle. Die Kopplung von SPS-I/O und SPS erfolgte mittels SPI-Bus.

Der Vergleich zwischen relativen Häufigkeiten auf der Versuchsanlage gemessener Antwortzeiten des in Abschnitt 6.2.1 diskutierten WLAN-Systems mit den analytisch bestimmten Wahrscheinlichkeiten findet sich in Abbildung 62. Deutlich zu erkennen ist, dass die Abweichungen erfreulich gering sind, wobei ähnliche Ergebnisse auch für andere Parameterkombinationen erzielt werden konnten (vgl. z.B. [Greifeneder und Frey, 2007a]).

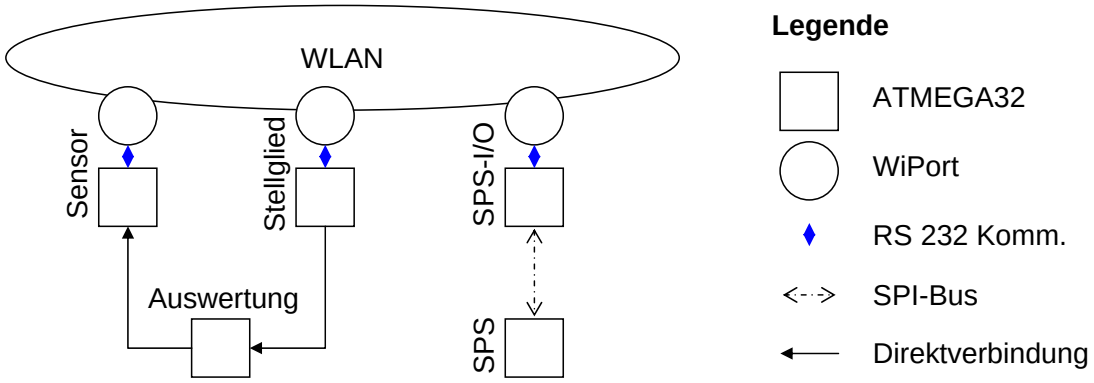

Abbildung 61: Struktureller Aufbau der Versuchsanlage

Die Anzahl von 1,3 Millionen Messpunkten war – wie in [Greifeneder et al., 2007] dargelegt – notwendig, um statistisch belastbare Ergebnisse zu erhalten, erforderte jedoch, dass der Messaufbau fünf Tage ununterbrochen betrieben wurde. Die Analyse mittels PRISM erfolgte hingegen in 17 Sekunden (stochastisches Until ohne absolute Zeitvariable).

Die hohe Übereinstimmung zwischen analytischen und messtechnisch ermittelten Werten bestätigt die erstellten Modelle und bildet die Grundlage einer vertrauenswürdigen Anwendung der vorgestellten Analysemethodik auf komplexere Strukturen und Fragestellungen.

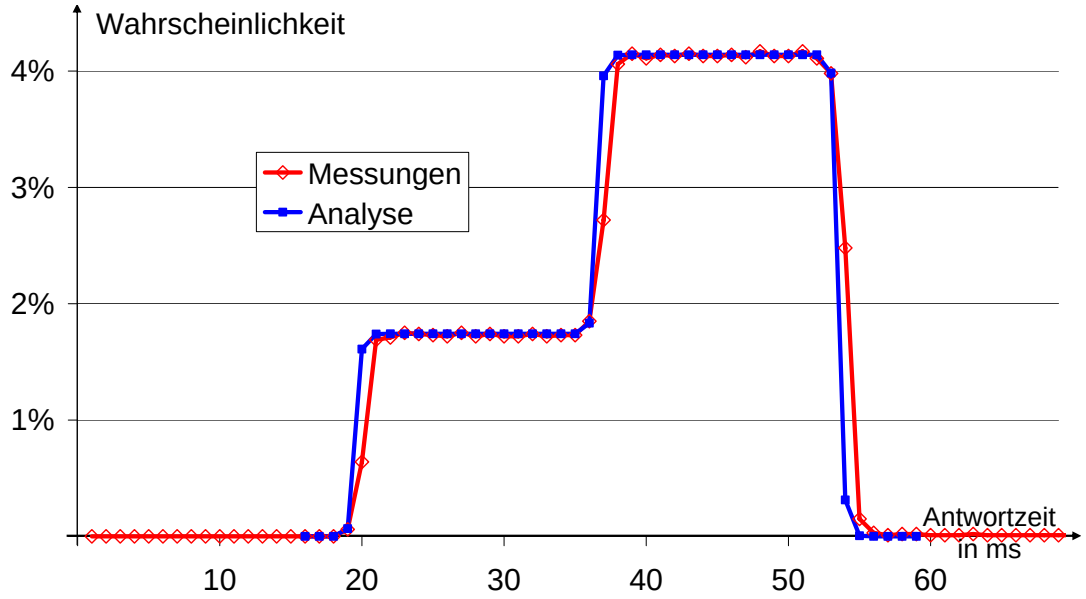

Abbildung 62: Relative Häufigkeiten von 1,3 Millionen auf der Versuchsanlage nach [Greifeneder et al., 2007] gemessenen Antwortzeiten im Vergleich mit den analytisch bestimmten Wahrscheinlichkeiten (SPS-Zyklus: 10 ms, SPS-I/O-Zyklus: 17 ms).

7 Zusammenfassung und Ausblick

Netzbasierte Automatisierungssysteme (NAS) erlauben völlig neuartige, flexible Architekturen in der Automatisierungstechnik. Die Sicherung von Antwortzeiten stellt sich dabei als eine wichtige Entwurfs- und Analyseaufgabe heraus. Von den literaturbekannten Analyseverfahren erfüllen lediglich die wahrscheinlichkeitsbasierte Modellverifikation (PMC) und die Simulation die NAS-spezifischen Anforderungen. Die Simulation scheitert jedoch i.d.R. an der mangelhaften statistischen Belastbarkeit der Ergebnisse. Obwohl zur effizienten Anwendung der PMC einige Vorarbeiten wie z.B. die Transformation auf den Ereigniseintritt notwendig sind, konnte PMC überzeugend zur Analyse temporaler auf Antwortzeiten abstrahierter Eigenschaften eingesetzt werden. Die diversen Möglichkeiten der Abstraktion ermöglichten es hierbei, stichhaltige Ergebnisse nach erfreulich kurzen Rechenzeiten zu erhalten. Der Vergleich mit Messdaten einer Laboranlage bestätigte die berechneten Verläufe.

In der vorliegenden Arbeit wurde erstmals ein vollständiger Ansatz zur formalen Analyse des Zeitverhaltens Netzbasierter Automatisierungssysteme vorgestellt. Dieser besteht aus zwei Aspekten. Zum einen bietet er die Definition einer für den Ingenieur leicht eingängigen Beschreibungssprache (DesLaNAS), welche zur Modellerstellung ebenso genutzt werden kann wie zur Spezifikation der zu analysierenden Prozesse und Eigenschaften. Zum anderen definiert er den für eine Verifikation mittels PMC notwendigen Transformationsprozess, welcher aus den folgenden Schritten besteht: Zunächst werden die in DesLaNAS erstellten Zusammenhänge in einer für NAS optimierten Automatendarstellung abgelegt. Auf dieser Basis findet dann die für die Untersuchung mittels PMC notwendige Diskretisierung und die Transformation in den PMC-Code statt. Im letzten Schritt erfolgt schließlich die Ausführung der Verifikation für die vom Ingenieur formulierte Fragestellung. Mit der neuen Methodik lassen sich nicht nur bestehende Systeme beurteilen, sondern sie ermöglicht auch die Offline-Beurteilung von Umkonfigurationen sowie Neuplanungen.

Mit dieser Arbeit ist die Basis für eine Reihe von möglichen Folgeprojekten gelegt. Hierzu gehört die umfangreiches Expertenwissen erfordernde Implementierung des reduktionsbasierten PMC-Ablaufs (vgl. Kapitel 3), inklusive der Erzeugung des notwendigen Signal-Trackings auf Basis generischer Templates. Weitere Vereinfachungen entstehen, wenn es gelingt, die optimale Wahl der Diskretisierung abzuleiten, z.B. aus der gewählten Eigenschaft oder einer vorgegebenen Ergebnisgenauigkeit. Gleiches gilt für die Entwicklung eines Verfahrens, mittels welchem die Anfangszustandsverteilung selbstläufig ermittelt werden kann. Schlussendlich ist eine Erweiterung des zeitkontinuierlichen Automaten auf kontinuierliche Eingangsgrößen ebenso interessant wie die Frage, ob sich die Verifikation analog der eingeführten Automatenmodelle modularisieren lässt.

Definitions- und Begriffsverzeichnis

Symbolverzeichnis

Symbol	Erläuterung	Seite
$\square$	Ende eines Beweises	
$\blacksquare$	Ende einer Definition	
$\diamondsuit$	abstraktes Nullelement der Dichtefunktion	
$\wedge,\ \vee$	Konjunktion, Disjunktion	
$\neg x$	Komplement von x	
$\exists,\ \in,\ \forall$	es existiert, Element von, für alle	
$\cap,\ \cup,\ \subseteq,\ \emptyset$	Vereinigung, Schnitt, Teil-, leere Menge	
∞	unendlich	
$\breve{}$	Schätzwert	
$\#$	Anzahl	
$\lvert Q\rvert$	Betrag, falls Q Vektor; Mächtigkeit, falls Q Menge	
2^Q	Potenzmenge	
$\mathtt{A_i}$	Zeitkontinuierlicher i-ter Teilautomat $\mathtt{A_i} = (\mathtt{Z_i}, \mathtt{Pr_i^0}, \mathtt{X_i}, \mathtt{X_i^0}, \mathtt{X_i^{act}}, \mathtt{W_i}, \mathtt{V_i}, \mathtt{C_i}, \mathtt{D_i}, \mathtt{Y_i}, \mathtt{L_i}, \mathtt{X_i^R})$	69
$\mathbf{A_i}$	Zeitdiskreter i-ter Teilautomat $\mathbf{A_i} = \left(\mathbf{Z_{p_i}}, \mathbf{z_{p_i}^0}, \mathbf{X_i}, \mathbf{x_i^0}, \mathbf{Z_{t_i}}, \mathbf{Q_i}, \mathbf{W_i}, \mathbf{V_i}, \mathbf{c_i}, \mathbf{p_i}, \mathbf{f_i}\right)$	76
$\mathtt{c}$	boolesche Funktion	
$\mathtt{d}$	Dichtefunktion	
$\mathtt{d}^*_{i_{k_j}}$	Übergangsbedingung des j-ten vom i_k-ten Zustand abgehenden Übergangs	
Δt	Zeitschrittweite	85
$\delta(a)$	Kurzschreibweise eines Dirac-Impulses an $x = a$	
$h(b)$	Heaviside-Funktion: $h(b) = \{0 \text{ falls } b < 0,\ 1 \text{ falls } b \geq 0\}$	
$\mathtt{GV}(a, b)$	Gleichverteilung: $\mathtt{GV}(a, b) = \frac{h(t-a+\varepsilon)\cdot h(b-t)}{b-a}$	68, 75

Fortsetzung auf nächster Seite

Symbol	Erläuterung	Seite
i	Zählvariable, Index des i-ten Teilautomaten	
$inc(x)$	increment: Erhöhe x um eins	79
m	in i_{k_m}: Anzahl der vom mit i_k referenzierten Zustand abgehenden Übergänge	
$\mathbb{N}$	Menge der natürlichen Zahlen, $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$	
ω	Pfad, $\lvert\omega\rvert$ Pfadlänge	35
`P=? [...]`	PCTL-Wahrscheinlichkeitsfunktion	26
$P(\mathbf{z}^\star, \omega)$	Wahrscheinlichkeit, um einen Zustand $\mathbf{z}^\star$ über einen Pfad ω zu erreichen	36
p	Wahrscheinlichkeit, allgemein	
$\mathtt{q}_{i_{k_{j_h}}}$	h-te stochastische Verzweigung von i_{k_j}	
`R=? [...]`	PCTL-Rewardfunktion	26
$R(\Omega(\mathbf{z}_0^\star))$	Kosten, um einen Zustand $\mathbf{z}^\star$ von $\mathbf{z}^0$ aus zu erreichen	37
r	r-ter diskreter Teilschritt	
$\mathbb{R}$	Menge der reellen Zahlen, $\mathbb{R}_0^+ = \{x \in \mathbb{R} \,\lvert\, x \geq 0\}$	
$res(x)$	reset: Rücksetzen von x auf $x = 0$	73, 79
t	Zeitvariable	
θ	Schätzvariable	91
`U`	PCTL-Until	26, 37
$\mathrm{Wsk}(\mathbf{z}^\star \lvert \mathbf{z}^0)$	Wahrscheinlichkeit, um einen Zustand $\mathbf{z}^\star$ von $\mathbf{z}^0$ kommend, zu erreichen	36
x	Uhrvariable, siehe Automaten	
$\mathtt{x}_{i_g}$	g-te Uhr des i-ten Teilautomaten	
z	Zustand, siehe Automaten	
$\mathtt{z}_{i_k}$	k-ter Zustand des i-ten Teilautomaten	

Abkürzungsverzeichnis

Abkürzung	Erläuterung
AT	Automatisierungstechnik
BDD	Binary Decision Diagrams
CAN	Controller Area Network
C^3	Computation, Communication, Control
CPN	Colored Petri-Net (Farbiges Petri-Netz)
CTL	Computation Tree Logic
CTMC	Continuous Time Markov Chain (kontinuierliche Markovkette)
DECS	Discrete Event Control System
DesLaNAS	Description Language for Networked Automation Systems
disk.	diskret
DTMC	Discrete Time Markov Chain (diskrete Markovkette)
GByte	Gigabyte
GHz	Gigahertz
GSPN	Generalized Stochastic Petri-Net (verallgemeinertes stochastisches Petri-Netz)
I/O	Input/Output (Ein-/Ausgang: E/A)
IP	Internetprotokoll
kByte	Kilobyte
kont.	kontinuierlich
LAN	Local Area Network
LTL	Linear Temporal Logic
MByte	Megabyte
MC	Model Checking (Modellverifikation)
MDP	Markov Decision Process

Fortsetzung auf nächster Seite

Abkürzung	Erläuterung
ms	Millisekunden
MTBDD	Multi-Terminal Binary Decision Diagrams
MTBF	mean time between failures
NAS	Netzbasiertes Automatisierungssystem (Networked Automation System)
NC	Network Calculus
NCS	Networked Control System
OPC	OLE for Process Control
PCTL	Probabilistic Computation Tree Logic
PMC	Probabilistic Model Checking (wahrscheinlichkeitsbasierte Modellverifikation)
PTCTL	Probabilistic Timed Computation Tree Logic
PTA	Probabilistic Timed Automaton (wahrscheinlichkeitsbasierter zeitbewerteter Automat)
QoC	Quality of Control
QoS	Quality of Service
SPS	Speicherprogrammierbare Steuerung
STPN	Stochastic Timed Petri-Net (stochastisches zeitbewertetes Petri-Netz)
TA	Timed Automaton (zeitbewerteter Automat)
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VPN	Virtual Private Network
WLAN	Wireless Local Area Network

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

Akers, Sheldon B.: *Binary decision diagrams.* IEEE Transactions on Computers, Band 27, Nr. 6, Seiten 509–516, 1978.

Alur, Rajeev, Costas Courcoubetis und David L. Dill: *Model-Checking for Probabilistic Real-Time Systems (Extended Abstract).* Automata, Languages and Programming, Seiten 115–126, 1991.

Alur, Rajeev, Costas Courcoubetis und David L. Dill: *Model-Checking in Dense Real-time.* Information and Computation, Band 104, Nr. 1, Seiten 2–34, 1993.

Alur, Rajeev und David L. Dill: *A theory of timed automata.* Theoretical Computer Science, Band 126, Nr. 2, Seiten 183–235, 1994.

Antolović, Marco, Kristen Acton, Naveen Kalappa, Siddharth Mantri, Jonathan Parrott, Jonathan E. Luntz, James R. Moyne und Dawn M. Tilbury: *PLC Communication using PROFINET: Experimental Results and Analysis.* Proc. 11^{th} IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA), Prag, Tschechien, 2006.

Antsaklis, Panos J. und John Baillieul: *Special Issue on Technology of Networked Control Systems.* Proceedings of the IEEE, Band 95, Nr. 1, Seite 5, 2007.

Artell, Tom, Hannu Koivisto und Jari Seppälä: *Quality of service in network-based automation.* Proc. 16^{th} IFAC World Congress, Prag, Tschechien, Elsevier, 2005.

Årzén, Karl-Erik und Anton Cervin: *Control and Embedded Computing: Survey of Research Directions.* Proc. 16^{th} IFAC World Congress, Prag, Tschechien, Elsevier, 2005.

Avižienis, Algirdas, Jean-Claude Laprie, Brian Randell und Carl Landwehr: *Basic Concepts and Taxonomy of Dependable and Secure Computing.* IEEE Transactions on Dependable and Secure Computing, Band 1, Nr. 1, Seiten 11–33, 2004.

Bahar, R. Iris, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo und Fabio Somenzi: *Algebraic decision diagrams and their applications.* IEEE/ACM Int. Conf. on Computer-aided design, IEEE Computer Society Press, Santa Clara, CA, Seiten 188–191, 1993.

Beauquier, Danièle: *On probabilistic timed automata.* Theoretical Computer Science, Elsevier, Band 292, Seiten 65–84, 2003.

Bérard, Béatrice, Michel Bidiot, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, Philippe Schnoebelen und Pierre McKenzie: *Systems and software verification, model-checking techniques and tools.* Springer, 2001.

Beyer, Dirk: *Formale Verifikation von Realzeit-Systemen mittels Cottbus Timed Automata.* Dissertation, Mensch & Buch, 2002.

Bohl, Joachim: *Experimentelle Bestimmung von Steuerungsgüte-Parametern am Beispiel einer Positioniersteuerung*. Studienarbeit SA022, JPA^2, Fachbereich EIT, Technische Universität Kaiserslautern, Deutschland, 2006.

Bolling, Beate und Ingo Wegner: *Improving the variable ordering of OBDDs is NP-complete*. IEEE Transactions on Computers, Band 45, Nr. 9, Seiten 993–1002, 1996.

Brand, Klaus-Peter: *IEC 61850: Standardisierter Informationsaustausch in Schaltanlagen und darüber hinaus*. ETH Kolloquium Aktuelle Probleme der Elektrotechnik, 2003.

Bretthauer, Georg: *Automatisierungstechnik – Quo vadis? Neun Thesen zur zukünftigen Entwicklung*. at – Automatisierungstechnik, Oldenbourg, Band 53, Nr. 4-5, Seiten 155–157, 2005.

Brück, Dag, Hilding Elmqvist, Hans Olsson und Sven Erik Mattsson: *Dymola for Multi-Engineering Modeling and Simulation*. Proc. 2^{nd} Int. Modelica Conference, Oberpfaffenhofen, Deutschland, Seiten 55:1–55:8, 2002.

Bryant, Randal E.: *Graph-based algorithms for Boolean function manipulation*. IEEE Transactions on Computers, Band 35, Nr. 8, Seiten 677–691, 1986.

Cassez, Frank und Kim G. Larsen: *On the Impressive Power of Stopwatches*. Proc. 11^{th} Int. Conf. Concurrency Theory (CONCUR), Catuscia Palamidessi (Editor), LNCS 1877, Springer, Seiten 138–152, 2000.

Cena, Gianluca, Ivan Cibrario Bertolotti und Andriano Valenzano: *Inexpensive tools for measuring Ethernet performance*. Proc. 2^{nd} IEEE Int. Conf. Industrial Informatics (INDIN), Berlin, Deutschland, Seiten 303–308, 2004.

Cena, Gianluca, Ivan Cibrario Bertolotti, Andriano Valenzano und Claudio Zunino: *Analysis of Response Times in 802.11 Industrial Networks*. Proc. 5^{th} IEEE Int. Conf. Industrial Informatics (INDIN), Wien, Österreich, Seiten 195–200, 2007.

Čerāns, Karlis: *Decidability of Bisimulation Equivalences for Parallel Timer Processes*. Proc. 4^{th} Int. Workshop Computer Aided Verification (CAV), Montreal, Canada, Gregor von Bochmann und David K. Probst (Editor), Lecture Notes in Computer Science 663, Springer, Seiten 302–315, 1992.

Clarke, Edmund M. und E. Allen Emerson: *Design and synthesis of synchronization skeletons using brachning time temporal logic*. Logic of Programs: Workshop, Yorktown Heights, NY, LNCS 131, Springer, 1981.

Clarke, Edmund M., Ansgar Fehnker, Sumit Kumar Jha und Helmut Veith: *Temporal Logic Model Checking*. Handbook of Networked and Embedded Control Systems, Dimitrios Hristu-Varsakelis und William S. Levine (Editoren), Birkhäuser, Seiten 539–558, 2005.

Clarke, Edmund M., Orna Grumberg und Doron A. Peled: *Model Checking*. MIT Press, Cambridge, 1999.

Clarke, Edmund M., Kenneth L. McMillan, Xudong Zhao, Masahiro Fujita und Jie Yang: *Spectral transforms for large Boolean functions with applications to technology mapping*. Formal Methods in System Design, Springer Netherlands, Band 10, Nummer 2–3, Seiten 137-148; ursprünglich veröffentlicht als Proc. 30th Int. Conf. Design automation, Dallas, Texas, ACM Press, Seiten 54–60, 1993.

Clarke, Edmund M., Kenneth L. McMillan, Xudong Zhao, Masahiro Fujita und Jie Yang: *Multi-terminal binary decision diagrams: An efficient data structure for matrix representation*. Formal Methods in System Design, Band 10, Nr. 2-3, Seiten 149–169, 1997.

Cook, Stephen A.: *The complexity of theorem-proving procedures*. Proc. 3rd annual ACM symposium on Theory of computing, Seiten 151–158, 1971.

Denis, Bruno, Silvain Ruel, Jean-Marc Faure, Gaëlle Marsal und Georg Frey: *Measuring the Impact of Vertical Integration on Response Times in Ethernet Fieldbusses*. Proc. Emerging Technologies and Factory Automation (ETFA), Patras, Griechenland, 2007.

Dingle, Nicholas J., Peter G. Harrison und William J. Knottenbelt: *Response time densities in generalised stochastic petri net models*. Proc. 3^{rd} Int. Workshop on Software and Performance (WOSP), Rom, Italien, ACM Press, Seiten 46–54, 2002.

Egea-Lopez, Esteban, Alejandro Martinez-Sala, Javier Vales-Alonso, Joan Garcia-Haro und Josemaria Malgosa-Sanahuja: *Wireless communications deployment in industry: a review of issues, options and technologies*. Computers in Industry, Elsevier, Band 56, Nr. 1, Seiten 29–53, 2005.

Enders, Reinhard, Thomas Filkorn und Dirk Taubner: *Generating BDDs for symbolic model checking in CCS*. Proc. 3rd Int. Workshop Computer Aided Verification (CAV), Aalborg, Dänemark, LNCS, Band 575, Springer, Seiten 203–213, 1991.

Ermont, Jérôme, Jean-Luc Scharbarg und Christian Fraboul: *Worst-case analysis of a mixed CAN/Switched Ethernet architecture*. Proc. Int. Conf. Real-Time and Network Systems (RTNS), Poitiers, Frankreich, Seiten 45–54, 2006.

Felser, Max: *Real-Time Ethernet – Industry Prospective*. Proceedings of the IEEE, Band 93, Nr. 6, Seiten 1118–1128, 2005.

Fritsch, Carsten, Jan H. Richter, Jan Lunze und Thomas Steffen: *Rapid Control Prototyping mit MATLAB/Simulink und speicherprogrammierbaren Steuerungen*. atp (Automatisierungstechnische Praxis), Oldenbourg, Band 48, Seiten 54–62, 2006.

Garey, Michael R. und David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

Glaser, Armin: *Anforderungen der Sicherheitstechnik an Ethernet-basierende Kommunikationssysteme.* atp – Automatisierungstechnische Praxis, Oldenbourg, Band 49, Nr. 3, Seiten 48–54, 2007.

Gottheit, Christian: *Experimentelle Bestimmung von Verzögerungen in verteilten Automatisierungssystemen.* Studienarbeit SA017, JPA^2, Fachbereich EIT, Technische Universität Kaiserslautern, Deutschland, 2006.

Greifeneder, Jürgen, Alexander Bödcher, Markus Trapp, Oliver Gabel und Mario Trapp: *Human Centered Manufacturing – Szenario „Man-u-Faktur 2012“.* Berichte des Forschungsschwerpunkts Ambient Intelligence Nr. 1 (http://kluedo.ub.uni-kl.de/frontdoor.php?source_opus=1652), Universität Kaiserslautern, Deutschland, 2003.

Greifeneder, Jürgen und Georg Frey: *Optimizing Quality of Control in Networked Automation Systems using Probabilistic Models.* Proc. 10$^{\text{th}}$ IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA), Catania, Italien, Band 1, Seiten 1065–1068, 2005.

Greifeneder, Jürgen und Georg Frey: *Determination of Delay Times in Failure Afflicted Networked Automation Systems using Probabilistic Model Checking.* Proc. 6$^{\text{th}}$ IEEE Int. Workshop on Factory Communication Systems (WFCS), Turin, Italien, Seiten 263–272, 2006a.

Greifeneder, Jürgen und Georg Frey: *Optimizing Quality of Control in Networked Automation Systems using Probabilistic Models.* Proc. 11$^{\text{th}}$ IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA), Prag, Tschechien, Seiten 372–379, 2006b.

Greifeneder, Jürgen und Georg Frey: *Probabilistic Hybrid Automata with Variable Step Width Applied to the Analysis of Networked Automation Systems.* Proc. 3$^{\text{rd}}$ IFAC Workshop on Discrete Event System Design (DESDes), Polen, Seiten 283–288, 2006c.

Greifeneder, Jürgen und Georg Frey: *Quantitative Analyse von Antwortzeiten in Netzbasierten Automatisierungssystemen.* Proc. SPS/IPC/DRIVES, Nürnberg, Deutschland, Seiten 243–252, 2006d.

Greifeneder, Jürgen und Georg Frey: *Analyse des Antwortzeitverhaltens netzbasierter Automatisierungssysteme.* atp – Automatisierungstechnische Praxis, Oldenbourg, Band 49, Nr. 10, Seiten 44–54, 2007a.

Greifeneder, Jürgen und Georg Frey: *Analyse netzbasierter Automatisierungssysteme.* Automation im gesamten Lebenszyklus (GMA-Kongress 2007), VDI-Berichte 1980, Baden-Baden, Deutschland, Seiten 23–33, 2007b.

Greifeneder, Jürgen und Georg Frey: *Wahrscheinlichkeitsbasierte Modellverifikation Netzbasierter Automatisierungssysteme.* at – Automatisierungstechnik, Oldenbourg, Band 55, Nr. 12, Seiten 624–633, 2007c.

Greifeneder, Jürgen, Liu Liu und Georg Frey: *Methoden zur Antwortzeitanalyse in vernetzten Automatisierungssystemen.* Proc. SPS/IPC/DRIVES, Nürnberg, Deutschland, Seiten 517–525, 2007.

Hansson, Hans und Bengt Jonsson: *A logic for reasoning about time and reliability.* Formal Aspects of Computing, Band 6, Nr. 5, Seiten 512–535, 1994.

Hartonas-Garmhausen, Vicky, Sergio Campos und Ed Clarke: *ProbVerus: Probabilistic Symbolic Model Checking.* Proc. 5^{th} Int. Workshop Formal Methods for Real-Time and Probabilistic Systems (AMAST), Springer, LNCS, Band 1601, Seiten 96–110, 1999.

Hédia, Belgacem Ben, Fabrice Jumel und Jean-Philippe Babau: *Formal Evaluation of Quality of Service for Data Acquisition Systems.* Proc. Forum of Specification and Design Languages, Lausanne, Schweiz, 2005.

Henzinger, Thomas A., Zohar Manna und Amir Pnueli: *What Good Are Digital Clocks?* Automata, Languages and Programming, Seiten 545–558, 1992.

Hérault, Thomas, Richard Lassaigne, Frédéric Magniette und Sylvain Peyronnet: *Approximate Probabilistic Model Checking.* Proc. 5^{th} Int. Conf. Verification, Model Checking, and Abstract Interpretation (VMCAI), Venedig, Italien, Springer, LNCS, Band 2937, Seiten 73–84, 2004.

Hermanns, Holger, Joost-Peter Katoen, Joachim Meyer-Kayser und Markus Siegle: *A tool for model-checking Markov Chains.* Int. Journal on Software for Technology Transfer (STTT), Band 4, Nr. 2, Seiten 153–172, 2003a.

Hermanns, Holger, Marta Z. Kwiatkowska, Gethin Norman, David Parker und Markus Siegle: *On the use of MTBDDs for performability analysis and verification of stochastic systems.* Journal of Logic and Algebraic Programming: Special Issue on Probabilistic Techniques for the Design and Analysis of Systems, Band 56, Nr. 1-2, Seiten 23–67, 2003b.

Hespanha, João P., Payam Naghshtabrizi und Yonggang Xu: *A Survey of Recent Results in Networked Control Systems.* Proceedings of the IEEE, Band 95, Nr. 1, Seiten 138–162, 2007.

Hughes, George E. und Max J. Cresswell: *Introduction to Modal Logic.* Methuen, 1977.

Humphrey, David: *Industrial Ethernet Networks the Digital Factory.* ARC strategies, 2006.

Irey, Philip, Brett L. Chappell, Robert W. Hott, David T. Marlow, Karen O'Donoghue und T.R. Plunkett: *Metrics, Methodologies, and Tools for Analyzing Network Fault Recovery Performance in Real-Time Distributed Systems.* Proc. Int. Parallel and Distributed Processing Symposium (IPDPS), Springer, LNCS, Band 1800, Seiten 1248–1257, 2000.

IST-Programme: *Report of Workshop on future and emerging control systems*. European Commission, 2000.

Jasperneite, Jürgen und Peter Neumann: *Switched Ethernet for Factory Communication*. Proc. 8^{th} IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA), Antibes, Frankreich, Seiten 205 – 212, 2001.

Jasperneite, Jürgen: *Analyse und Modellierung von Kommunikationslasten in der Fertigungstechnik*. at - Automatisierungstechnik, Oldenbourg, Band 49, Seiten 206 – 213, 2001.

Jasperneite, Jürgen: *Leistungsbewertung eines lokalen Netzwerkes mit Class-of-Service Unterstützung für die prozessnahe Echtzeitkommunikation*. Dissertation, Otto-von-Guericke Universität Magdeburg, Shaker, 2002.

Jensen, Henrik Ejersbo: *Model Checking Probabilistic Real Time Systems*. Proc. 7^{th} Nordic Workshop on Programming Theory, Göteborg, Schweden, Band 86, Seiten 247–261, 1996.

Juanole, Guy und Gérard Mouney: *QoS in real time distributed systems and process control applications*. Proc. Workshop Networked Control Systems & Fault Tolerant Control (NeCST), Ajaccio, Frankreich, Seiten 11–18, 2005.

Katoen, Joost-Peter, Maneesh Khattri und Ivan S. Zapreev: *A Markov Reward Model Checker*. Proc. 2^{nd} Quantitative Evaluation of Systems (QEST), Turin, Italien, IEEE Computer Society, Seiten 243–244, 2005.

Kofman, Ernesto, Marcelo Lapadula und Esteban Pagliero: *PowerDEVS: A DEVS-Based Environment for Hybrid System Modeling and Simulation*. Technical Report LSD0306, LSD, Universidad Nacional de Rosario, Argentinien, 2003.

Kopetz, Hermann: *Time-triggered real-time computing*. Annual Reviews in Control, Elsevier, Band 27, Seiten 3–13, 2003.

Kwiatkowska, Marta Z.: *Model Checking for Probability and Time: From Theory to Practice*. Invited Paper, Proc. 18^{th} Annual IEEE Symposium on Logic in Computer Science (LICS), Ottawa, Kanada, IEEE Computer Society Press, Seiten 351–360, 2003.

Kwiatkowska, Marta Z., Gethin Norman und David Parker: *PRISM: Probabilistic Symbolic Model Checker*. Proc. 12^{th} Int. Conf. Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS), Springer, LNCS, Band 2324, Seiten 200–204, 2002.

Kwiatkowska, Marta Z., Gethin Norman und David Parker: *Modelling and Verification of Probabilistic Systems*. Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems, Jan Morgenstern (Editor). CRM Monograph Series, 23. American Mathematical Society, Seiten 93–215, 2004a.

Kwiatkowska, Marta Z., Gethin Norman und David Parker: *Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach*. International Journal on Software Tools for Technology Transfer (STTT), Band 6, Nr. 2, Seiten 128–142, 2004b.

Kwiatkowska, Marta Z., Gethin Norman, Jeremy Sproston und Farn Wang: *Symbolic Model Checking for Probabilistic Timed Automata*. Information and Computation, Band 205, Nr. 7, Seiten 1027–1077, 2007.

Laroussinie, François und Jeremy Sproston: *Model Checking Durational Probabilistic Systems*. Proc. 8^{th} Int. Conf. Foundations of Software Science and Computational Structures (FOSSACS), Edinburgh, UK, LNCS, Springer, Band 3441, Seiten 140–154, 2005.

Lee, Chung-Yee: *Representation of switching circuits by binary-decision programs*. Bell System Technical Journal, Band 38, Seiten 985–999, 1959.

Liu, Liu und Georg Frey: *Simulation Approach for Evaluating Response Times in Networked Automation Systems*. Proc. 12^{th} IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA), Patras, Griechenland, 2007.

Lunze, Jan: *Ereignisdiskrete Systeme. Modellierung und Analyse dynamischer Systeme mit Automaten, Markovketten und Petrinetzen*. Oldenbourg Verlag München Wien, 2006.

Maciej, Rosól: *Network Control Under CAN Bus Technology*. Proc. 12^{th} IEEE Int. Conf. Methods and Models in Automation and Robotics (MMAR), Miedzyzdroje, Polen, Seiten 557–562, 2006.

Marsal, Gaëlle: *Evaluation of time performances of Ethernet-based Automation Systems by simulation of high-level Petri Nets*. Dissertation, TU Kaiserslautern, Shaker, 2007.

Marsal, Gaëlle, Bruno Denis, Jean-Marc Faure und Georg Frey: *Evaluation of Response Time in Ethernet-based Automation Systems*. Proc. 11^{th} IEEE Int. Conf. Emerging Technology and Factory Automation (ETFA), Prag, Tschechien, Seiten 380–387, 2006.

Miorandi, Daniele und Stefano Vitturi: *Performance Analysis of Producer/Consumer Protocols over IEEE 802.11 Wireless Links*. Proc. IEEE Int. Workshop on Factory Communication Systems (WFCS), Wien, Österreich, 2004.

Moyne, James R. und Dawn M. Tilbury: *The Emergence of Industrial Control Networks in Manufacturing Control, Diagnostics, and Safety Data*. Proceedings of the IEEE, Band 95, Nr. 1, Seiten 29–47, 2007.

Murray, Richard M., Karl J. Åström, Stephen P. Boyd, Roger W. Brockett und Gunter Stein: *Future directions in control in an information-rich world*. Report of the NSF-Panel on Future Directions in Control, Dynamics and Systems, IEEE Control Systems Magazine, Band 23, Nr. 2, Seiten 20–33, 2003.

Ohlin, Martin, Dan Henriksson und Anton Cervin: *TrueTime 1.4 - Reference Manual*. Dep. of Automatic Control, Lund University, Schweden, 2006.

Oldenkamp, Marcel: *Probabilistic model checking, A comparison of tools*. Masterthesis, University of Twente, Niederlande, 2007.

Parker, David: *Implementation of Symbolic Model Checking for Probabilistic Systems*. Dissertation, University of Birmingham, England, 2002.

Parker, David, Gethin Norman und Marta R. Kwiatkowska: *PRISM 2.1.dev4 User's Guide*. The University of Birmingham. Die jeweils aktuelle Version ist verfügbar unter: http://www.prismmodelchecker.org/manual/, 2005.

Parrott, Jonathan T., James R. Moyne und Dawn M. Tilbury: *Experimental Determination of Network Quality of Service in Ethernet: UDP, OPC, and VPN*. Proc. of American Control Conference (ACC), Minneapolis, USA, 2006.

Pereira, Nuno, Eduardo Tovar und Luis Miguel Pinho: *Timeliness in COTS factory-floor distributed systems: what role for simulation?* Proc. IEEE Workshop on Factory Communication Systems (WFCS), Wien, Österreich, Seiten 13–21, 2004.

Pinho, Luís Miguel und Francisco Vasques: *Timing Analysis of Reliable Real-Time Communication in CAN Networks*. Proc. 13^{th} Euromicro Conf. on Real-Time Systems, Delft, Niederlande, 2000.

Pnueli, Amir: *The temporal logic of programs*. Proc. 18^{th} IEEE Symp. Foundations of Computer Science (FOCS), Providence, RI, USA, Seiten 46–57, 1977.

Quielle, Jean-Pierre und Joseph Sifakis: *Specification and verification of timed circuits*. Proc. 5^{th} Int. Symposium on Programming, Seiten 337–350, 1981.

Ratzer, Anne V., Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen und Kurt Jensen: *CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets*. Proc. 24^{th} Int. Conf. Applications and Theory of Petri Nets (ICATPN), Eindhoven, Niederlande, Springer, LNCS, Band 2679, Seiten 450–462, 2003.

Rauchhaupt, Lutz, Elke Hintze und André Gnad: *Über die Bewertung der Zuverlässigkeit industrieller Funklösungen*. atp – Automatisierungstechnische Praxis, Oldenbourg, Band 49, Nr. 3, Seiten 38–47, 2007.

Ridouard, Frédéric, Jean-Luc Scharbarg und Christian Fraboul: *Stochastic network calculus for end-to-end delays distribution evaluation on an avionics switched Ethernet*. Proc. 5^{th} IEEE Int. Conf. Industrial Informatics (INDIN), Wien, Österreich, Seiten 559–564, 2007.

Schmitt, Jens B. und Frank A. Zdarsky: *The DISCO network calculator: a toolbox for worst case analysis*. Proc. 1^{st} Int. Conf. on Performance evaluation methodolgies and tools, Artikel Nr. 8, Pisa, Italien, ACM Press, 2006.

Schnieder, Eckehard und Karl Heinz Kraft: *Berechnung von Reaktionszeiten in Steuerungssystemen mit verteilten Elementen.* Siemens Forschungs- und Entwicklungsberichte, Band 9, Nr. 6, Seiten 325–329, 1980.

Schnieder, Eckehard und Roman Slovak: *PROFUND: Ein integrativer Ansatz zum Entwurf verlässlicher Automatisierungssysteme.* Automation im gesamten Lebenszyklus (GMA-Kongress), VDI-Berichte, Band 1980, Seiten 227–236, 2007.

Sen, Koushik, Mahesh Viswanathan und Gul Agha: *On Statistical Model Checking of Stochastic Systems.* Proc. 17th Conf. Computer Aided Verification (CAV), Edinburgh, Schottland, Springer, LNCS, Band 3576, Seiten 266–280, 2005a.

Sen, Koushik, Mahesh Viswanathan und Gul Agha: *VESTA: A Statistical Model-checker and Analyzer for Probabilistic Systems.* Proc. 2nd Quantitative Evaluation of Systems (QEST), Turin, Italien, IEEE Computer Society, Seiten 251–252, 2005b.

Somenzi, Fabio: *CUDD: Colorado University decision diagram package.* Public software, Colorado University, Boulder (http://vlsi.colorado.edu/∼fabio/), 1997.

Song, Yeqiong, Anis Koubfia und François Simonot: *Switched Ethernet for real-time industrial communication: Modelling and message Buffering delay evaluation.* Proc. 4^{th} IEEE Int. Workshop Factory Communication Systems (WFCS), Vasteras, Sweden, Seiten 27–35, 2002.

Soucek, Stefan und Thilo Sauter: *Quality of service concerns in IP-based control systems.* IEEE Trans. on Industrial Electronics, Band 51, Nr. 6, Seiten 1249 – 1258, 2004.

Stanczyk, Jaroslaw und Andrzej Obuchowicz: *The max-plus algebra approach to the prototyping of concurrent processes.* Proc. 9^{th} IEEE Int. Conf. Methods and Models in Automation and Robotics (MMAR), Miedzyzdroje, Polen, Band 2, Seiten 857–862, 2003.

Stöhr, Markus: *Netzwerkanalyse mittels TrueTime und Jitterbug.* Studienarbeit SA026, JPA^2, Fachbereich EIT, Technische Universität Kaiserslautern, Deutschland, 2007.

Tani, Seiichiro, Kiyoharu Hamaguchi und Shuzo Yajima: *The Complexity of the Optimal Variable Ordering Problems of Shared Binary Decision Diagrams.* Proc. 4th Int. Symposium on Algorithms and Computation, Hong Kong, 1993.

Varga, András: *The OMNeT++ discrete event simulation system.* Proc. European Simulation Multiconference (ESM), Prag, Tschechien, 2001.

Vojnović, Milan und Jean-Yves Boudec: *Stochastic Analysis of Some Expedited Forwarding Networks.* Proc. 21^{st} IEEE Annual Joint Conf. Computer and Communications Societies (INFOCOM), New York, U.S.A., Band 2, Seiten 1004–1013, 2002.

Vyatkin, Valeriy und Hans-Michael Hanisch: *Verification of distributed control systems in intelligent manufacturing.* Journal of Intelligent Manufacturing, Springer, Band 14, Nr. 1, 2003.

Walsh, Gregory C., Hong Ye und Linda Bushnell: *Stability analysis of networked control systems*. IEEE Transactions on Control Systems Technology, Band 10, Nr. 3, Seiten 438–446, 2002.

Wang, Farn: *Formal verification of timed systems: a survey and perspective*. Proceedings of the IEEE, Band 92, Nr. 8, Seiten 1283–1305, 2004.

Witsch, Daniel und Birgit Vogel-Heuser: *Techniken zur effizienten Verifikation von Echtzeitsystemen durch Model-Checking*. Proc. 9^{th} Symposium Entwurf komplexer Automatisierungssysteme (EKA), Braunschweig, Deutschland, Eckehard Schnieder (Hrsg), Seiten 117–129, 2006.

Younes, Håkan L. S.: *Ymer: A Statistical Model Checker*. Proc. 17^{th} Int. Conf. of Computer Aided Verification (CAV), Edinburgh, Schottland, Springer, LNCS, Band 3576, Seiten 429–433, 2005.

Younes, Håkan L. S., Marta Z. Kwiatkowska, Gethin Norman und David Parker: *Numerical vs. Statistical Probabilistic Model Checking*. International Journal on Software Tools for Technology Transfer (STTT), Band 8, Nr. 3, Seiten 216–228, 2006.

Zhang, Qizhi, Yunze Cai, Danying Gu und Weidong Zhang: *Determine the maximum closed-loop control delay in switched industrial Ethernet using network calculus*. Proc. American Control Conference (ACC), Minneapolis, Minnesota USA, 2006.

Zimmermann, Armin und Jan Trowitzsch: *Eine quantitative Untersuchung des European Train Control System mit UML State Machines*. Proc. 9. Symp. zu Entwurf komplexer Automatisierungssysteme (EKA), Braunschweig, Deutschland, Seiten 271–304, 2006.

Normenverzeichnis

DIN 40041 *Zuverlässigkeit; Begriffe*. Deutsches Institut für Normung e.V., 1990.

DIN EN IEC 61508 *Funktionale Sicherheit elektrischer/elektronischer/programmierbar elektronischer sicherheitsbezogener Systeme*. Deutsches Institut für Normung e.V.

- Teil 4 Begriffe und Abkürzungen, 2002.

- Teil 6 Beispielhafte Vorgehensweise zur Bestimmung von Hardware-Ausfällen, Abschnitt B3: Wahrscheinlichkeit eines Ausfalls je Stunde (für die Betriebsart mit hoher Anforderungsrate oder kontinuierlicher Anforderung), 2001.

IEC 61703 *Mathematical expressions for reliability, availability, maintainability and maintenance support terms*. International Electrotechnical Commission, 2001.

IEC 61850 *Communication networks and systems in substations*. International Electrotechnical Commission, 2007.

IEEE 802 *Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Network – Specific Requirements*.

- Teil 3 *Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. Institute of Electrical and Electronics Engineers, 2005.

- Teil 11 *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. Institute of Electrical and Electronics Engineers, 1999.

IFIP WG-10.4 *Dependable Computing and Fault Tolerance*. http://www.dependability.org, International Federation for Information Processing, 1988.