Analyse des Antwortverhaltens vernetzter Automatisierungssysteme mittels Probabilistic Model Checking

Jürgen Greifeneder und Georg Frey
Technische Universität Kaiserslautern, FB EIT, JPA²
Erwin-Schrödinger-Straße 12
D-67663 Kaiserslautern
Tel.: +49 (0)631 205-4461
Fax: +49 (0)631 205-4462
E-Mail: {greifeneder,frey}@eit.uni-kl.de

Abstract: Die Einführung nicht-deterministischer Netzwerke im Bereich der Automatisierungstechnik führt zu neuen Fragestellung bei der Analyse dieser Systeme. Dies liegt insbesondere am Antwortverhalten von Automatisierungssystemen. Abhängig von den zu analysierenden Eigenschaften sind Signale, die um mehr als eine kritische Zeitspanne verzögert eintreffen gleichzusetzen, mit Signalen, die durch Komponentenausfälle verloren gegangen sind. Bei der Analyse der Zuverlässigkeit müssen zusätzlich zu weiteren, ausfallfähigen Komponenten auch durch das Netzwerk verursachte Verzögerungen in die Betrachtungen aufgenommen werden. Zur Untersuchung von Systemen mit nicht-deterministischem Komponentenverhalten erweist sich das Probabilistic Model Checking (PMC) als viel versprechender Ansatz. In diesem Beitrag wird ein Modellierungsansatz für obige Systeme unter Verwendung von PMC vorgestellt und gezeigt, wie damit Zuverlässigkeitsanalysen durchgeführt werden können. Anhand eines Anwendungsbeispiels wird demonstriert, wie die aus der Überlagerung verschiedener zyklischer Prozesse resultierenden Verzögerungen mittels PMC ermittelt werden können.

Stichworte: Zuverlässigkeitsanalyse, Netzwerke, Probabilistic Model Checking, Antwortzeiten

1 Einleitung

Die Zuverlässigkeit eines Systems wurde vom 10. Technischen Committee (Computer Systems Technology) der International Federation for Information Processing (IFIP) in der Arbeitsgruppe vier (Dependable Computing and Fault Tolerance) als "the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers" [http://www.dependability.org] definiert. Die Zuverlässigkeit eines Systems wird gleich gesetzt mit einer ganzen Menge von Anforderungen, wie z.B. Ausfallsicherheit, Verfügbarkeit, Fehlertoleranz, Robustheit, Sicherheit und Gefahrlosigkeit.

Im Bereich der Verlässlichkeitsanalyse von Automatisierungssystemen (AT-Systemen) ist es notwendig das aus Hard- und Software bestehende System zu betrachten. Diese Herausforderung

Greifeneder, J.; Frey, G.: *Analyse des Antwortverhaltens vernetzter Automatisierungssysteme mittels Probabilistic Model Checking.* Proceedings of the 9th Conference on Entwurf komplexer Automatisierungssysteme (EKA 2006), Braunschweig, Germany, pp.131-151, May 2006.

wird durch die Ausbreitung von netzwerkbasierten Systemen erschwert, da in diesem Fall der Einfluss des Netzwerkes zusätzlich zu dem der einzelnen Komponenten für die Zuverlässigkeitsbestimmung Berücksichtigung finden muss, denn das Netzwerk bringt nicht nur (anzunehmender Maßen nicht-deterministische) Verzögerungen in das System ein, sondern auch weitere zu berücksichtigende Arbeitszyklen und ausfallgefährdete Komponenten.

Modellierung und Analyse von AT-Systemen erfordert nicht nur Detailwissen über die funktionalen Aspekte der Komponenten, sondern auch über ihr Echtzeitverhalten und mögliche Ausfälle. Insbesondere ist es erforderlich die jeweils erzeugten Verzögerungen zu kennen, da die meisten Controller[1] Informationen über die sie betreffende Prozesshardware (hiermit sind Sensoren und Aktuatoren, bzw. allgemein Eingänge und Ausgänge (I/O) des Systems, gemeint) innerhalb beschränkter Zeitfenster benötigen, um korrekt arbeiten zu können. Daraus lassen sich Eigenschaftsspezifikationen wie z.B. „Die Reaktion auf einen Signalwechsel erfolgt innerhalb von 0,2 s“ oder „ein lediglich 5ms lang anstehendes Signal wird durch den Controller erkannt.“

In herkömmlichen Anlagen, in welchen ein einzelner Controller direkt mit den zugehörigen I/Os verbunden ist, folgen hieraus hauptsächlich Anforderungen an die Zykluszeit des Controllers. Dennoch ergibt sich schon im Falle der einfachen zyklischen Verarbeitung eines Signals durch eine Speicherprogrammierbare Steuerung (SPS) ein – im einfachsten Falle gleich verteiltes – Spektrum von Verzögerungszeiten im Reaktionsverhalten. Besitzen sowohl SPS als auch deren I/O-Karte eigene, voneinander unabhängige (nicht exakt synchronisierte) Zeitzyklen, wird das Reaktionsverhalten bereits deutlich komplexer. Finden sich im System mehrere Prozesse, ist von einer exakten Synchronisation derselben nur in seltenen Fällen auszugehen. Zugriffskonflikte und Wartezeiten führen zu weiteren nicht-deterministischen Verzögerungen.

Im Falle von verteilten Systeme mit einem oder mehreren Controllern, die über ein Netzwerk mit ihren I/Os kommunizieren, verschärft sich das Problem insofern, als die durch das Netzwerk hervorgerufenen Verzögerungen berücksichtigt werden müssen, welche in der Regel durch eine Verteilungsfunktion (und nicht durch eine einfache Konstante) dargestellt werden können. Bereits bei einem einzelnen Controller mit über Ethernet angeschlossenem I/O-Modul erhält man für die Verzögerungszeit des Gesamtsystems Verteilungen, die ohne formale Analysen nur noch schwer zu ermitteln sind [GreFr2005].

Gerade in Bezug auf die Qualität eines Produktes spielt die möglichst genaue Analyse von Verzögerungen eine entscheidende Rolle, da eine geringere Spannweite der Verzögerungen (z.B. bei der Positionierung eines Bohrers oder Fahrbettes) direkten Einfluss auf die Fertigungsgenauigkeit und somit auf die Endqualität (bzw. der Notwendigkeit zur Nachpositionierung) haben kann.

---

[1] Im Folgenden wird der Begriff „Controller“ als Überbegriff für Steuerungen und Regelungen verwendet.

Greifeneder, J.; Frey, G.: *Analyse des Antwortverhaltens vernetzter Automatisierungssysteme mittels Probabilistic Model Checking.* Proceedings of the 9th Conference on Entwurf komplexer Automatisierungssysteme (EKA 2006), Braunschweig, Germany, pp.131-151, May 2006.

Die Verwendung von worst-case-Ansätzen führt leider meist zu nicht mehr erfüllbaren Anforderungsprofilen an die verwendete Hardware, weshalb so genannte probabilistic bounds für die Formulierung der Anforderungen verwendet werden. Obige Beispiele würden nun also lauten: „Die Reaktion auf einen Signalwechsel erfolgt mit mindestens 99,9%-ger Sicherheit innerhalb von 0,2 s“ und „ein lediglich 5ms lang anstehendes Signal wird in 85% der Fälle durch den Controller erkannt.“

Will man ein komplexes System auf Eigenschaften dieser Art hin überprüfen, scheidet die Simulation aus zwei Gründen heraus aus: Erstens benötigen wahrscheinlichkeitsbehaftete Modelle sehr lange Simulationszeiten und zweitens liefern selbige keine hinreichend exakten Lösungen. Im Falle einer detaillierten Zuverlässigkeitsanalyse (statt des einfachen Performance-Checks) verschärft sich dieses Problem insofern, als die Ausfallmöglichkeiten der Komponenten (inklusive des Netzwerkes) bei der Analyse ebenso berücksichtigt werden müssen, wie kleinste Verschiebungen der Zykluszeiten zueinander. Dabei weist das System Zykluszeiten auf, die im Vergleich zu den erwartbaren Fehlereintrittszeiten (mean times between failures – MTBFs) geradezu winzig erscheinen.

Die formale Beschreibung derartiger Systeme führt auf Modellierungsformen, welche Zeiten, stochastische Verteilungen und bedingte Eintrittswahrscheinlichkeiten aufweisen. Auf Basis probabilistischer zeitbewerteter Automaten (probabilistic timed automata – PTA) lassen sich unter Verwendung der Probabilistic Computing Tree Logic (PCTL) die bedingten Eintrittswahrscheinlichkeiten mittels Probabilistic Model Checking (PMC [KwNoPa2004]) ermitteln. Im vorliegenden Beitrag wurde PMC zur Analyse von Verzögerungszeiten in netzwerkbasierten AT-Systemen verwendet.

1.1 Anforderungsklassifikation

Während sich klassische kontinuierliche Regelungsprobleme auf eine generische Problembeschreibungsebene (Ausregeln der Störgrößen, Einregeln von Regelgrößen) bringen lassen – und somit generische Gütemaße angebbar sind – ist dies im Falle von Steuerungen für ereignisdiskrete Systeme (discrete event systems control – DESC) nicht möglich. 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.

Grundsätzlich kann zwar zwischen zeitabhängigen und zeitunabhängigen Anforderungen unterschieden werden, allerdings gibt es kaum Anwendungsbeispiele, bei denen beliebige Verzögerungen möglich oder zulässig sind, so dass sich die Menge der zeitunabhängigen Anforderungen als Untergruppe der zeitabhängigen Anforderungen erweist. Damit lässt sich die Menge der zu diskutierenden Eigenschaftstemplates auf die folgenden Fälle abbilden:

Greifeneder, J.; Frey, G.: *Analyse des Antwortverhaltens vernetzter Automatisierungssysteme mittels Probabilistic Model Checking.* Proceedings of the 9th Conference on Entwurf komplexer Automatisierungssysteme (EKA 2006), Braunschweig, Germany, pp.131-151, May 2006.

1. Maximal- und Minimalzeiten. Beispiel hierfür sind die Reaktionszeit, welche ab der Aktivierung des Notausknopfes vergeht sowie das früheste zulässige Öffnen einer Klemme nach Auslösen eines Befehls.
2. Verteilungen: Mit welcher Wahrscheinlichkeit kann auf ein Signal nach x Sekunden reagiert werden?
3. Differenzen, Abstände: Zeitdifferenz mit der zwei aufeinander folgende Datenpakete eintreffen oder die Frage nach der Wahrscheinlichkeit einer korrekten Reihenfolge beim Eintreffen dieser Datenpakete.

1.2 Signaltypen

Möchte man Aussagen über Ausfälle in redundanten Datennetzwerken machen, reicht die alleinige Kenntnis der Wahrscheinlichkeitsfunktion eines bestimmten Ausfalles nicht aus. Ebenso wichtig ist es, die Zeiten zu berücksichtigen, die notwendig sind, um einen solchen Ausfall zu erkennen sowie, um ihn 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.

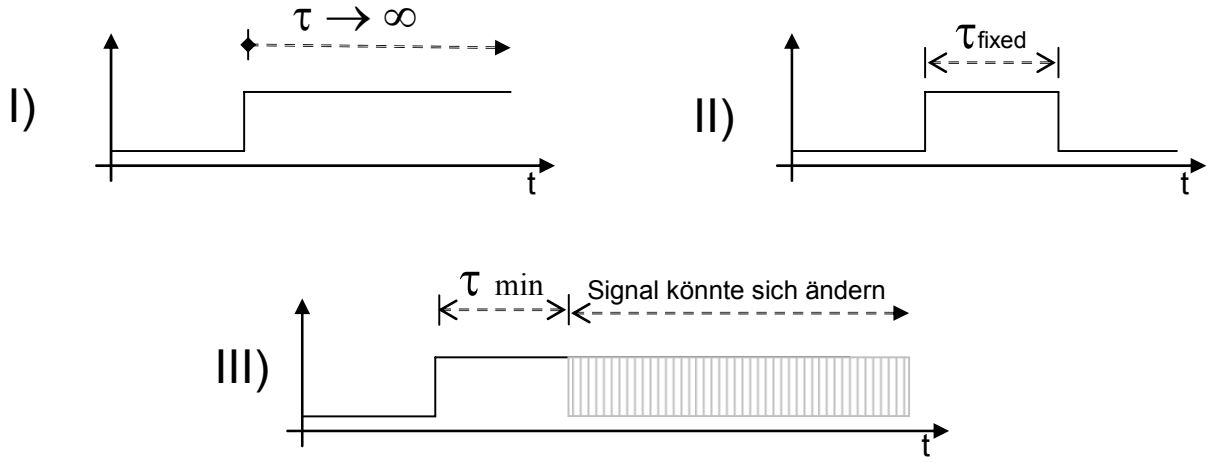

Abbildung 1: Drei verschiedene Typen von Signalen

In einem automatisierungstechnischen System lassen sich die folgenden drei Signaltypen unterscheiden (Abbildung 1):

1. 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?”

2. Ein Signal, das nur für eine bestimmte Zeit τ_{fixed} aktiv ist. Ist diese Zeit abgelaufen, ist die Information unwiderruflich verloren, falls sie nicht vorher ausgelesen worden ist. Die Frage in diesem Fall lautet daher: „Wird das Signal überhaupt erkannt“?

3. Die Kombination der vorherigen beiden Fälle. Hierunter fällt z.B. ein Signal, welches sich mit der 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 der Signalwechsel erkannt wird?

Die im Folgenden betrachteten netzwerkbasierten AT-Systeme bestehen aus einem oder mehreren Controllern (hier im Sinne von Hardware, z.B. SPS), welche über ein aus I/O-Modulen und Switches gebildetes Netzwerk mit dem Prozess verbunden sind. Die Komponenten solcher Systeme tauschen Signale und Informationen auf der Basis von TCP/IP aus. Da manche dieser Komponenten mehr als (nur) einmal im System vorkommen, sollte ein Modell des Systems in einfacher Weise skalierbar sein. Außerdem macht es Sinn, die unterschiedlichen Komponenten unabhängig voneinander zu modellieren und sie zum Gesamtsystem zu verbinden.

2 Modellierungsmethode

Zur Modellierung der einzelnen Komponenten wurden PTA verwendet. Durch eine Diskretisierung der Zeitachse ergibt sich die Chance, nicht-deterministische Entscheidungssituationen auf eine Menge diskreter Zeitschritte zu elimieren und im Zuge dessen sämtliche relevanten Entscheidungen (Transitionen) synchron durchzuführen. Im Rückschluss bedeutet dies, dass das Modell nicht die exakten Auftretenszeiten eines Ereignisses wiedergibt, sondern lediglich die Tatsache, dass es innerhalb des letzten Zeitschrittes aufgetreten ist. In manchen Systemen ist es jedoch nicht möglich, eine als optimal angesehene Zeitschrittweite in der Weise zu bestimmen, dass jedes beteiligte Teilsystem genau eine Aktion ausführen kann. Wählt man die Zeitschrittweite so, dass sie kürzer ist, als es der kürze zeitliche Abstand zwischen zwei Systemänderungen ist, weist die Größe des Modells nämlich die Tendenz zu exponentiellem Wachtum auf.

In den folgenden Unterkapiteln wird zunächst näher auf das Probabilistic Model Checking eingegangen und darauf aufbauend die gewählte Systemmodellierung vorgestellt. Dieser folgt eine Diskussion der Anfangszustandsproblematik und eine Vorstellung der Eigenschaftsformulierung in PCTL. Den Abschluss des Kapitels bildet die Modulkonzeption. In den jeweiligen Unterkapiteln wird auch kurz auf eine entsprechende Codierung im verwendeten Model-Checker PRISM eingegangen. PRISM [KwNoPa2002, http://www.cs.bham.ac.uk/~dxp/prism/] wurde an der University of Birmingham implementiert.

2.1 Probabilistic Model Checking

Beim klassischen Model Checking [BeBiFi2001] wird zunächst – unter Verwendung einer formalisierten Beschreibungssprache – ein Modell des Systems erstellt. Unabhängig hiervon werden die zu überprüfenden Eigenschaften in einer formalen Logik formuliert. Diese beiden formalen Beschreibungen finden Eingang in den Model Checking-Algorithmus welcher prüft, ob das Systemmodell die geforderten Eigenschaften erfüllt (Abbildung 2). Systemmodell und Eigenschaften werden hierbei unabhängig voneinander behandelt. Eine Änderung des Systems beeinflusst daher lediglich das Systemmodell, nicht jedoch die formale Eigenschaftsbeschreibung und umgekehrt.

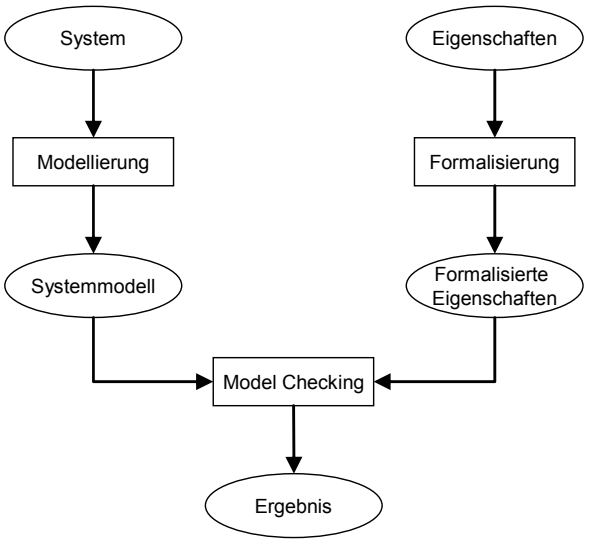

Abbildung 2: Workflow von klassischem Model Checking

Der größte Vorteil der Verwendung von Model Checking liegt in der kompletten Abdeckung aller möglichen Evolutionspfade eines Systems (statt sich auf eine durch Simulation oder Testverfahren i.A. gegebene Untermenge zu beschränken). Den größten Nachteil bildet die Tatsache, dass der zu bildende Zustandsraum die unangenehme Eigenschaft aufweist, sehr stark mit der Komplexität des Modells anzuwachsen (Problem der Zustandsexplosion).

Die Zustandsexplosion 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.

Selbstverständlich gilt dies auch für PMC, allerdings müssen die dort verwendeten Abbruchbedingung (vgl. Abschnitt 2.4) nicht wiederkehrenden Charakter aufweisen. Bei zyklischen Problemstellungen, bei denen ein Ereignis über der Zeit wiederholt vorkommt kann die Wahrscheinlichkeit des Eintritts dieses Ereignisses als geometrische Reihe beschrieben werden, welche für große Zeiten ($t \rightarrow \infty$) gegen eins konvergiert. Dies bedeutet, dass in zyklischen Systemen die

Wahrscheinlichkeit für den Eintritt eines (mit der Wiederholung erneut auftretenden) Ereignisses Eins oder Null ist – wahr oder falsch. 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.

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. Diese Anforderungen verändern den Entwurfsprozess jedoch grundlegend: Die Trennung zwischen Modell und Eigenschaften muss aufgegeben und ein neuer formaler Entwurfsprozess gefunden werden (Abbildung 3).

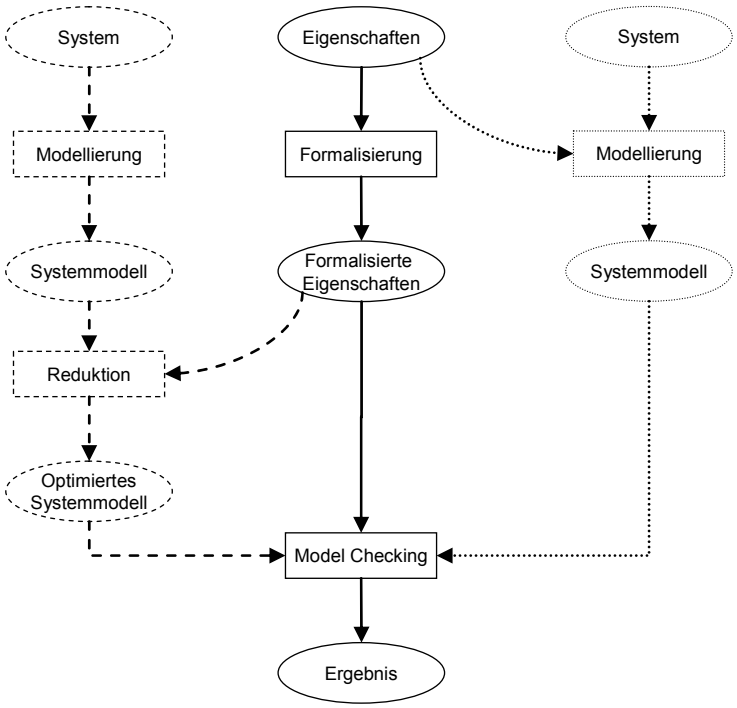

Abbildung 3: Workflow von probabilistischem Model Checking (PMC)

Während die Eigenschaften wie zuvor in Abbildung 2 formuliert werden – in Abbildung 3 findet sich dies nun in der Mitte der Abbildung wieder – muss der Entwurfsprozess des Modells angepasst werden. Die erste Möglichkeit dies zu tun ist in gestrichelten Linien auf der linken Seite von Abbildung 3 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 3 in gepunkteten Linien 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 Ingenieur, das Modell für jede zu prüfende Eigenschaft neu zu erstellen. Im Rahmen dieser Arbeit wurde der zwei-

te (gepunktete) Ansatz verwendet, da es nicht möglich ist, einen sinnvollen Reduktionsalgorithmus vorzuschlagen ohne ein sehr genaues Verständnis dessen zu besitzen, was ein als optimal anzusehendes Modell ausmacht. Im weiteren Verlauf des beschriebenen Projektes ist es jedoch durchaus angedacht, die bis dahin vorhandene Erfahrung im Umgang mit entsprechenden Modulen in einen sinnvollen Reduktionsansatz umzusetzen und somit den im gestrichelten, linken Ast beschriebenen Ansatz zu wählen.

Während bisher nur abstrakt von den Systemmodellen und Eigenschaftsbeschreibungen gesprochen wurde, werden diese in den folgenden Teilkapiteln kurz vorgestellt und die damit zusammenhängenden Probleme und Herausforderungen näher beleuchtet.

2.2 Systemmodellierung

Das Systemmodell wird unter Verwendung endlicher Automatenmodelle mit Erweiterungen für Zeiten und wahrscheinlichkeitsbedingtes Verhalten erstellt. In diesem Teilkapitel wird das Modell in mehreren Schritten erklärt. Ein endlicher Automat (Abbildung 4) besteht aus durch konditionale Verbindungen verknüpften Zuständen.

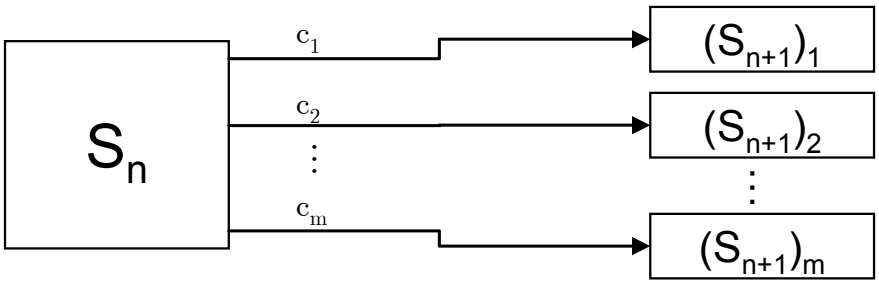

Abbildung 4: Endlicher Automat

Besitzt ein Zustand S_n eines Automaten m mögliche Nachfolgezustände, welche mittels der Transitionen t_1 bis t_m mit dem Zustand S_n verbunden seien, so gilt das Folgende, wobei die zu t_1 bis t_m gehörigen Schaltbedingungen mit c_1 bis c_m bezeichnet wurden.

Es gibt keine zwei Schaltbedingungen, die zur selben Zeit aktiv wären:

$$\bigwedge_{i,j\in[1..m];i\neq j} c_i \wedge c_j = false \tag{1}$$

Die Vereinigung aller Schaltbedingungen muss den gesamten Zustandsraum abdecken:

$$\bigvee_{\forall i\in[1..m]} c_i = true \tag{2}$$

Zusammen bedeutet dies, dass immer genau eine Schaltbedingung innerhalb eines Moduls aktiv sein muss.

Aus Gründen der Zeitskalierung sollte eine Schaltbedingung nur aktiv werden, wenn eine zuvor bestimmte Zeit abgelaufen ist. Dies lässt sich unter Verwendung von zeitbewerten Automaten (Timed Automata – TA) erreichen (Abbildung 5a).

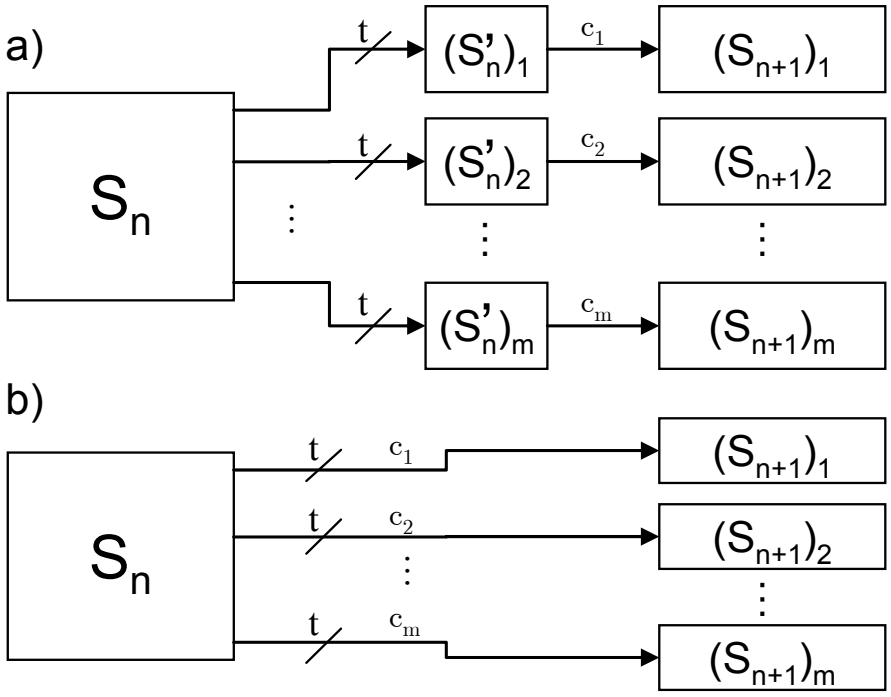

Abbildung 5: Zeitbewerteter Automat (Timed Automaton)

Ist die Zeit t abgelaufen, werden alle mit dem Synchronisationssignal *t* (sync-Signal) beschrifteten Transitionen aktiv. Für diese Arbeit gelten die folgenden beiden Einschränkungen: Erstens gibt es lediglich einen Zeittakt auf den synchronisiert wird (die Verwendung mehrerer verschiedener sync-Signale ist bei Verwendung eines allgemeinen getakteten Automaten durchaus möglich). Zweitens wird die Schaltbedingung der nachfolgenden Transition in dem Moment ausgewertet, in dem die sync-Bedingung aktiv geworden ist. Unter Verwendung dieser Vereinfachungen macht es Sinn, die mit S'_n gekennzeichneten Zwischenzustände wegzulassen und die in Abbildung 5b gezeigte graphische Darstellung zu verwenden.

Der nächste Erweiterungsschritt führt direkt zu den probabilistischen zeitbewerteten Automaten (Probabilistic Timed Automata – PTA) (Abbildung 6a). Im Fall wahrscheinlichkeitsgewichteter Zustandsübergänge, wird jeder ausgehenden Transition eine Wahrscheinlichkeit $p_i \in [0..1]$, mit

$$\sum_{\forall i \in [1..k]} p_i = 1 \tag{3}$$

zugeordnet. In anderen Worten wird der bisherige, deterministische Automat um eine nichtdeterministische Auswahl erweitert, deren Transitionen mit den Wahrscheinlichkeiten p_i gewich-

tet werden[2]. Da auch die Zwischenzustände S"$_n$ transienter Natur sind, wird im Folgenden lediglich die in Abbildung 6b gezeigte vereinfachte Darstellung verwendet.

Anmerkung: Aus Vereinfachungsgründen werden Wahrscheinlichkeiten die gleich eins sind und Bedingungen die immer zu wahr evaluiert würden nicht im Graphen dargestellt.

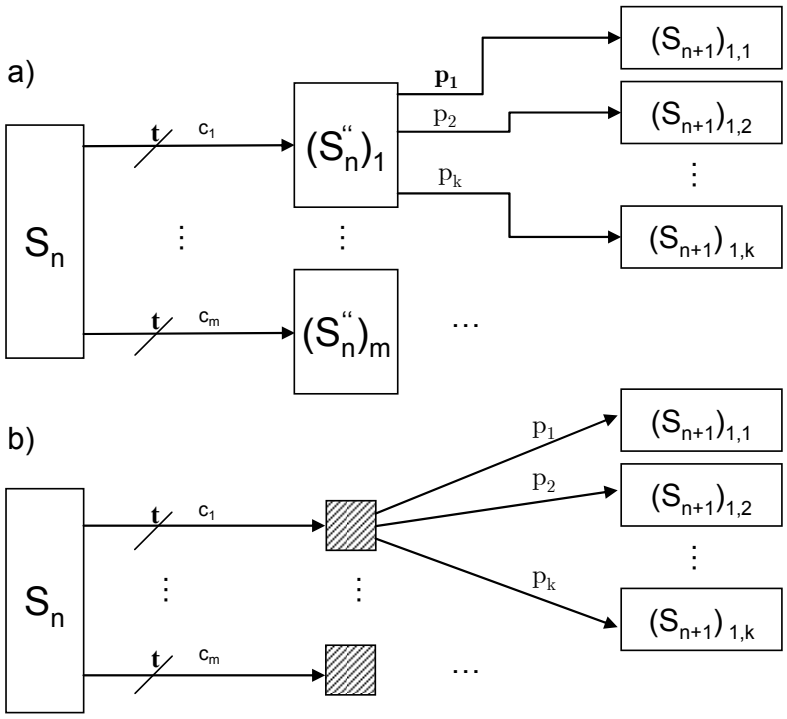

Abbildung 6: Probabilistischer zeitbewerteter Automat (PTA).

Der PRISM-Code des Systemmodells unterliegt der folgenden Semantik:

[Pr] conditions → assignments;

[Pr] conditions → p1:assignments + p2:assignments + … ;

[Pr] stellt dabei das bereits in Abschnitt 2.2 diskutierte Synchronisationssignal dar. Wird es weggelassen, geht der Determinismus der Sequenz verloren. Unter conditions versteht man Aussagen, die aus einer oder mehrerer Schaltbedingungen c_i (oder deren negierter Partner) zusammengesetzt sind. Assignments sind zulässige Wertzuweisungen für eine oder mehrere Variablen.

[2] Der Unterschied zwischen einer nicht-deterministischen Auswahl zwischen zwei Pfaden und einer mit je 50% Wahrscheinlichkeit angegebenen stochastischen Auswahl besteht darin, dass in PMC im zweiten Fall beide Pfade mit 50% Wahrscheinlichkeit berücksichtigt würden, während sich im ersten Fall streng genommen zwei eigenständige Automaten ergeben, einmal nur mit der einen und das andere mal nur mit der anderen Auswahlmöglichkeit. Da somit die Entscheidung zwischen zwei Pfaden weggefallen ist und nur noch je ein Weg existiert wird die zugehörige Wahrscheinlichkeit dieses einen Zustandsüberganges mit 100% angesetzt werden und nicht mit 50%, wie im zweiten Fall.

Diese Wertzuweisungen können aus Konstanten oder Funktionen beliebiger Variablenwerte (vom letzten Zeitpunkt vor dem Schaltvorgang) zusammengesetzt sein. Last but not least, stellen $p_1, p_2 \ldots p_k$ die Wahrscheinlichkeitswerte dar.

Der vorgestellte Ansatz kann als konform zur in [AlCoDi1991] gegebenen PTA-Definition angesehen werden, falls eine dichte Zeitschrittweite Verwendung findet. Nicht kompatibel ist die gewählte Modellierungsform allerdings zu PTA-Definitionen, wie sie z.B. bei [BeBiFi2001] Verwendung finden. Bei dieser Definitionsform wird der Zeitpunkt der Transitionsaktivierung durch das Eintreten eines Zustandes bestimmt, wohingegen die Zeit lediglich zur Bestimmung der dann aktiven Transition genutzt wird. Unter diesem Gesichtspunkt ist die dortige Definition also genau konträr zu der im Rahmen dieser Arbeit vorgestellten. Streng genommen findet durch die vorgenommene Reduktion eine Abbildung des PTA auf eine zeitdiskrete Markow-Kette statt.

Darüber hinaus unterscheidet sich der vorgestellte Ansatz von Ansätzen, wie sie z.B. von [AlCoDi1991], [KwNoPa2004] verwendet werden durch den Zeitpunkt zu dem die Zeitdiskretisierung vorgenommen wird. Während dieser Schritt im gewählten Ansatz bereits zum Zeitpunkt der Modellierung durchgeführt wird, findet er sich in obigen Ansätzen erst nach der Modellbeschreibung und unmittelbar vor dem Übergang zum Model Checking. Man spricht in diesem Zusammenhang von der Bildung einer endlichen Automaten-Quotienten-Darstellungsform. Unter den hierbei angewandten Verfahren finden sich z.B. digital clocks [HenNi1992] und region graphs [AlCoDi1993].

Eine Verwendung kontinuierlicher Markow-Ketten oder stochastischer Petrinetze [DaAl2005] konnte nicht stattfinden, da diese statt festen Schaltzeitpunkten exponentiell verteilte Übergangsraten verwenden. Gerade in den betrachteten technischen Anlagen treten jedoch feste Zeitintervalle auf, z.B. bei Zykluszeiten von Prozessen oder „time-outs“ bei Übertragungen.

2.3 Anfangszustand

Einer der wichtigsten Unterschiede zwischen Simulation und Model Checking besteht darin, dass Model Checking mit Sicherheit alle möglichen Zustände des untersuchten Systems mindestens einmal erreichen wird, wohingegen es bei einer Simulation möglich ist, dass selbst nach einer langen Simulationszeit nicht alle Zustände erreicht worden sind. Demgegenüber wäre es unsinnig, im zyklischen System einen Model Checking Algorithmus mehr als für einen Zyklus auszuführen, wenn es möglich ist, die Menge der Anfangszustände so zu wählen, dass jeder mögliche Zustand im ersten Zyklus erreicht werden kann. Hierdurch wird es deutlich einfacher, die zuvor geforderte Eigenschaft des Modells – zu terminieren, nachdem der gesuchte Zustand eingetreten ist – umzusetzen.

Eine korrekte (und vollständige) Wahl der Anfangszustände ist schwer. Verfügt das System über mindestens zwei Teilprozesse (Module), die nicht mit der selben Zykluszeit ablaufen, ergibt sich eine größere Anzahl möglicher zeitlicher Verschiebungen zwischen den Anfangspunkten dieser

beiden Zyklen, insbesondere, wenn diese nicht gleichzeitig gestartet wurden. Im besten Fall liegen lediglich zwei driftende Module vor, wobei die möglichen Zeitverschiebungen als gleichverteilt angenommen werden können (d.h. müssen). In diesem Fall lässt sich die Anfangsverteilung dadurch erstellen, dass in einem der beiden Module vor den ersten Zustand ein Vorprozess geschaltet wird, welcher einen „gleichverteilten Initialzustand“ erzeugt. Dies ist in Abbildung 7 gezeigt.

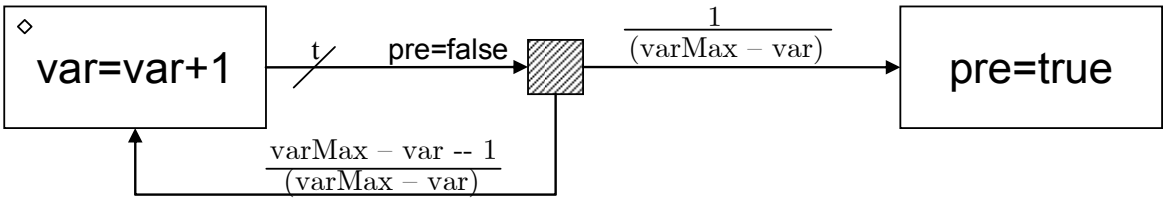

Abbildung 7: Gleichverteilter Anfangszustand

Ausgehend vom mit ◊ gekennzeichneten Zustand führt dieser Automat die folgende Programmschleife aus: Solange *pre=false* ist, setze *pre* mit einer Wahrscheinlichkeit von $p = (varMax - var)^{-1}$ auf *true*, andernfalls erhöhe *var* um eins. Im ersten Schritt wird also *pre* mit einer Wahrscheinlichkeit von $(varMax)^{-1}$ auf *true* gesetzt (und somit *var*=0 beibehalten). Im zweiten Schritt ist *var*=1, der rechte Zustand in Abbildung 7 würde also mit einer Wahrscheinlichkeit von $(varMax - 1)^{-1}$ auf *true* gesetzt. Da es jedoch nur mit einer Wahrscheinlichkeit von $(varMax - 1) \cdot (varMax)^{-1}$ zu diesem zweiten Schritt kam, ist die Wahrscheinlichkeit, dass *var* am Ende den Wert *var*=1 angenommen hat zu

$$\frac{varMax - 1}{varMax} \cdot \frac{1}{varMax - 1} = \frac{1}{varMax} \tag{4}$$

Am Ende dieses Vorprozesses weißt die Variable *var* also die Werte 0 bis *varMax*-1 jeweils mit einer Wahrscheinlichkeit von *varMax* auf. *varMax* ist hierbei ein frei wählbarer Parameter, der die Anzahl vom Automaten zu repräsentierenden Zustände angibt.

Wie man leicht sieht, ist die Schrittanzahl bei diesem Verfahren nicht für alle Anfangskombinationen gleich groß ist. Dies hat zur Folge, dass bei Aufgabenstellungen, bei denen es auf die Schrittanzahl ankommt, obigem Automaten ein Prozess nachgelagert werden muss, der die bis zum Ende des Vorprozesses durchzuführenden Schritte für alle Anfangszustände auf *varMax* hoch zählt. Ein ähnliches Problem ergibt sich, wenn mehrere solcher Anfangszustandsautomaten Verwendung finden sollen. In diesem Fall muss sichergestellt werden, dass der Anfangszustandsdeterminierungsprozess erst als abgeschlossen gilt, wenn alle Einzelautomaten den mit *var=true* gekennzeichneten Zustand erreicht haben. Hierfür muss obiger Automat um eine Schleife im rechten Zustand ergänzt werden.

Der zugehörige PRISM-Code lautet wie folgt:

[P] !pre&var<varMax → 1/(varMax-var):(pre=true) + (var-varMax-1)/(var-varMax):(var'=var+1);

[P] pre&(!pre2|!pre3|…|!pren) → true;

In diesem Algorithmus repräsentiert [P] den sync-operator, *pre* ist eine Binärvariable welche anzeigt, ob der Initialprozess abgeschlossen ist oder nicht. !*pre* ist die Negation von *pre*. *pre2*… *pren* sind die zu *pre* analogen Variablen anderer (synchronisierter) Anfangszustandsmodule. Die zweite Zeile dient damit, wie zuvor erläutert, dazu, auf die anderen Anfangszustandsmodule zu „warten" und so ein Verklemmen des Gesamtautomatens zu verhindern.

Selbstverständlich wäre es möglich, die Wertezuweisung in einem einzelnen Schritt zu erreichen, in dem der Variable *var* mit $(varMax)^{-1}$ die 0, mit $(varMax)^{-1}$ die 1 usw. zugewiesen würde. Dieses Verfahren hat jedoch leider den Nachteil, dass es statischen Charakter aufweist, d.h. der Programmcode für jeden Wert von *varMax* entsprechend kodiert werden muss.

2.4 Formulierung der Eigenschaften in PCTL

PMC verwendet zur Spezifizierung von Systemeigenschaften eine Erweiterung von CTL (PCTL – Probabilistic Computation Tree Logic) [HanJo1999]. Typischerweise bestehen diese Eigenschaften aus Elementaroperatoren oder logischen Verknüpfungen der Variablen im Modell. Streng genommen wird zwischen Zustands- und Pfadkonstrukten unterschieden, was jedoch für die weitere Betrachtung nicht von Interesse ist. Normalerweise liefert PCTL nur boolsche Ergebnisse. Allerdings ist es oftmals nützlich den genauen Wahrscheinlichkeitswert statt nur eines boolschen Ergebnisses zu erhalten, weshalb viele Implementierungen diese Funktion unterstützen.

In PRISM können beispielsweise Eigenschaften in der folgenden Form angegeben werden:

P=? [*expr1* U *expr2*]

P=? bedeutet, dass der Wahrscheinlichkeitswert zurückgegeben werden soll. *expr1* und *expr2* sind logische Aussagen welche Boolsche Ergebnisse haben. Obiger Befehl liest 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 – *expr*2 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! Die einfachste Weise, diesen Operator zu verwenden, besteht darin, *expr1* durch *true* zu ersetzen und lediglich mit dem zweiten Prädikat (*expr2*) zu arbeiten:

P=? [*true* U *expr2*]

Ausgehend von einem gegebenen Systemmodell und einer Menge gewünschter Eigenschaften, muss der PRISM-Code erzeugt werden. Dabei ist es wichtig, dass der Algorithmus so bald als sinnvoll möglich terminiert, wobei zwei Fälle unterschieden werden können:

a) Ist die Dauer einer bestimmten Aktion gesucht – z.B. die Verzögerung – ist es erforderlich, abzuwarten, bis diese Aktion beendet oder (zur praktischen Begrenzung) eine maximale Zeit vergangen ist. Dies kann mit dem folgenden Operator erreicht werden:

P=? [true U Runtime=Lf]

Lf ist ein Parameter – PRISM überprüft die Eigenschaft für jeden innerhalb eines vorgegebenen Intervalls liegenden Wert von *Lf* – und *Runtime* ist eine Variable, welche durch das Modell gesetzt wird. Sobald die überwachte Aktion eingetreten ist, wird der aktuelle Wert des Laufzeitzählers in diese Variable geschrieben. Es wäre zwar möglich, direkt auf die Lautzeitzählervariable zu testen, aber in diesem Falle würde sich als Ergebnis das Integral der Verteilungsfunktion (ausgewertet von *Lf* bis *unendlich*) ergeben.

b) Soll die Wahrscheinlichkeit bestimmt werden, mit der ein bestimmtes Ereignis (überhaupt) eintritt, so darf die Berechnung nicht durch den Eintritt des Ereignis abgebrochen werden. Diese Aufgabe kann nur dann korrekt gelöst werden, wenn die Testperiode zeitlich beschränkt ist, namentlich jener Zeit innerhalb welcher das Ereignis genau einmal (jedoch kein zweites Mal!) eintreten kann. Zur Lösung dieses Problems ist es wichtig alle möglichen Anfangszustände im ersten Zyklus (vgl. Abschnitt 2.3) abzudecken und den Automaten nach Beendigung dieses ersten Zyklus zu terminieren.

2.5 Modulkonzeption

Betrachtet man die bisher aufgestellen Charakteristiken des zu entwerfenden Modells und strukturiert anhand ihrer den Code, so finden sich drei Klassen von Modulen:

1. Grundfunktionen
2. Architekturabhängige Verknüpfungen
3. Signaltracking

Die Gruppe der Grundfunktionen zeichnet sich dadurch aus, dass sie unabhängig vom jeweiligen Problem oder den gesuchten Eigenschaften implementiert werden kann. Hierzu gehören z.B. die Grundeigenschaften eines Sensormoduls seinen Signalwert auf Anfrage auszugeben, aber auch der periodische Durchlauf einer Speicherprogrammierbaren Steuerung (SPS). Diese Grundelemente müssen dann natürlich je nach Problemstellung durch weitere Module ergänzt werden, insbesondere muss die SPS wissen, welche I/O-Module sie abfragen muss und wie sie – falls dies explizit modelliert werden soll – die empfangenen Werte verarbeiten soll. Da diese Spezifikationen (zu denen z.B. auch gehört, welche I/O-Karte an welchen Switch angeschlossen ist) als Gemeinsamkeit ihre architekturabhängige Natur haben, wurden sie zur zweiten Gruppe zusammengefasst. Die dritte Gruppe bezieht sich auf die Überprüfung der Eigenschaft. Hierzu gehören

alle Module, die die Erfüllung der gesuchten Eigenschaft mitverfolgen. Diese sind erforderlich, da es in der gewählten Abfragesyntax nicht möglich ist, Abfolgen von Zuständen durch PCTL verfolgen zu lassen.

Im Idealfall ist es möglich, diese drei Modulklassen unabhängig voneinander zu gestalten. 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.

3 Anwendungsbeispiel

Die Struktur des zu diskutierenden Anwendungsbeispiels ist in Abbildung 8 dargestellt. In diesem Beispiel konzentriert sich die Untersuchung auf die Überlagerung verschiedener zyklischer Prozesse. Eine Erweiterung hin zur Zuverlässigkeitsanalyse durch die Berücksichtigung von Komponentenausfällen ist analog zu [GreFr2005] möglich

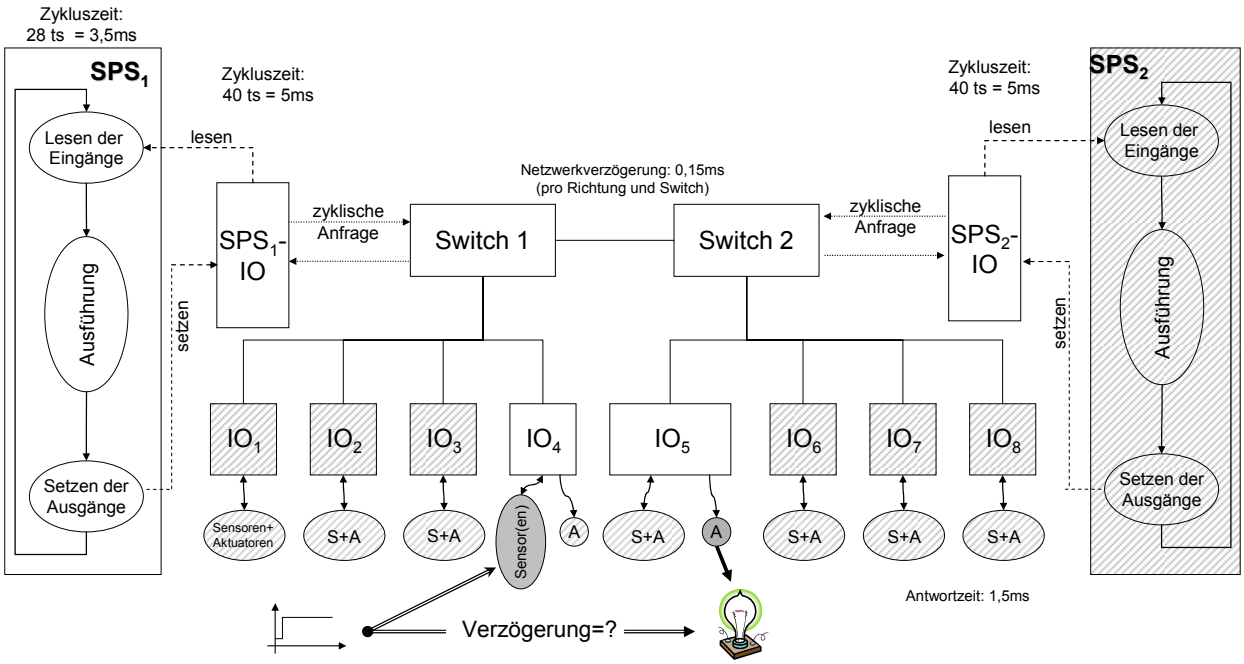

Abbildung 8: Anwendungsbeispiel

Das System besteht aus zwei SPSen, die je einen Controller beinhalten und mit einer Zykluszeit von 3,5ms arbeiten. Jede der beiden SPSen hat eine angeschlossene IO-Karte, welche einen unabhängigen Zyklus von 5ms aufweist. Während SPS_1 an Switch 1 angeschlossen ist, ist SPS_2 direkt mit Switch 2 verbunden. Das System weist 8 digitale IO-Karten auf, an denen jeweils ein oder mehrere Sensoren und Aktuatoren angeschlossen sind. Die IO-Karten 1 bis 4 sind hierbei mit dem ersten Switch, die IO-Karten 5 bis 8 mit dem zweiten Switch sowie die beiden Switches untereinander verbunden. Die linke SPS benötigt für ihre Ausführung die Sensorinformationen

der IO-Karten 1 bis 5, während die rechte SPS die Informationen der IO-Karten 4 bis 8 benötigt und somit diese zyklisch abfrägt. Hierbei ist zu beachten, dass in dem zugrunde liegenden Client-Server-Protokoll die SPS (=Client) Anfragen an die I/O-Module (=Server) sendet und letztere nur auf Anfragen reagieren, d.h. von sich aus keine Informationen versenden.

Um ein Datenpaket zu versenden entstehen die folgenden Verzögerungen: 0,375ms, um ein Paket zu ver- bzw. entpacken und 0,06ms pro zu passierendem Switch. Da ein sinnvolles Warteschlangenkonzept für die Switche noch nicht entworfen wurde (vgl. hierzu Kapitel 4), wurde vereinfachend davon ausgegangen, dass die Übertragung pro Switch 0,125ms erfordert. Erhält eine der digitalen I/O-Karten eine Anfrage, vergehen 1,5ms ehe die zugehörige Antwort wieder ins Netzwerk eingespeisst worden ist – allerdings nur dann, wenn die I/O-Karte gerade keine andere Anfrage abzuarbeiten hat. Dies kann für die I/O-Karten 4 und 5 der Fall sein, da beide von beiden SPSen Anfragen erhalten. Ist eine I/O-Karte beim Eintreffen einer Anfrage gerade beschäftigt, so muss die Anfrage warten, bis die Bearbeitung der vorherigen Anfrage abgeschlossen ist. Hierdurch entstehen weitere Verzögerungen.

Es wird angenommen, dass SPS_1 bei einem Signalwechsel an I/O-Board 4 einen Aktuatorausgang an I/O-Board 5 aktiviert. Das an I/O-Board 4 auftretende Signal habe Signaltyp-I-Charakteristik, d.h. dass Signal behält seinen Wert bei, bis es vom Controller erkannt, bzw. die zugehörige Aktion ausgeführt worden ist. Gesucht ist die Verteilung der Verzögerung, die sich zwischen dem Signalwechsel an IO_4 und dem Aktivieren des Aktuators an IO_5 ergibt.

Die SPS-IO-Karten fragen die I/O-Boards in numerisch aufsteigender Reihenfolge ab, d.h. erst 1, dann 2, 3, 4 und 5 bzw. 4, dann 5, 6, 7 und 8. Durch diese Annahme rechtfertigt sich die Vernachlässigung einer detaillierten Modellierung des Netzwerkes. Letzteres hat darüber hinaus zur Folge, dass die I/O-Boards 1 bis 3 und 6 bis 8 nur indirekt in Modell erscheinen, nicht aber als eigene Modulblöcke implementiert werden. Ähnliches gilt für die rechte SPS, deren Verhalten für die von Interesse befindliche Aufgabenstellung keine weitere Rolle spielt.

SPS, Verzögerungszeitmessung und die Kernmodule der I/O-Karten lassen sich als Modultyp 1 (vgl. Abschnitt 2.5) realisieren und sind somit problemunabhängig verwendbar. Darüber hinaus werden für die I/O-Karten noch spezielle Algorithmen benötigt, die die Verknüpfungsstrukturen beschreiben (Modultyp 2). Aufgrund des relativ einfach gehaltenen Beispiels lässt sich das komplette Signaltracking (Modultyp 3) in einem einzigen unabhängigen Modul realisieren. Ebenfalls von Modultyp 3 ist das Abbruchmodul. Die Synchronisation der einzelnen Module erfolgt über den gemeinsamen Zeitgeber, der den Zeitpunkt des Feuerns der jeweils aktiven Transition in jedem angeschlossenen Modul angibt. Des weiteren sind die Module auch durch gemeinsam verwendete Variablen verkoppelt. Dies ist verhältnismäßig einfach möglich, da auch eine nicht global definierte Variable in PRISM global sichtbar ist, eine Wertzuweisung allerdings nur in dem sie definierenden Modul möglich ist. Im Folgenden sind die einzelnen Module kurz vorgestellt.

Greifeneder, J.; Frey, G.: *Analyse des Antwortverhaltens vernetzter Automatisierungssysteme mittels Probabilistic Model Checking.* Proceedings of the 9th Conference on Entwurf komplexer Automatisierungssysteme (EKA 2006), Braunschweig, Germany, pp.131-151, May 2006.

3.1 Detaillierte Module

Das Modul der SPS sowie die Kernmodule der beiden SPS-I/O-Karten bestehen aus Ringzählern, die vom Initialwert bis zum Maximalwert hoch zählen, um dann bei 0 wieder zu beginnen (Abbildung 9). Wie in allen weiteren Diagrammen ist auch in Abbildung 9 die durch die im Modul Abbruch festzustellende Beendigung hervorgerufene Rücksetzung nicht dargestellt, da sie von jedem Zustand aus existiert und somit die Diagramme äußerst unübersichtlich werden lassen würde.

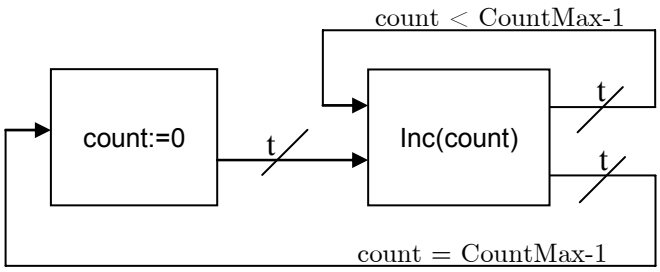

Abbildung 9: Ringzählers.

Für die Verzögerungszeitmessung wurde das selbe Modul verwendet, allerdings ohne Rücksetzung bei Erreichung der Maximalzeit (CountMax). In diesem Fall verbleibt der Zähler auf der Maximalposition.

Für die beiden SPS-I/O-Karten, als auch für die SPS selbst, wurden Initialzustandsautomaten realisiert, wie in Abschnitt 2.3 diskutiert (vgl. Abbildung 7).

Da das Netzwerk nicht modulmäßig abgebildet sondern lediglich durch eine konstante Zeitverschiebung berücksichtig wurde, müssen die für die I/O-Boards benötigten Eingangsmodule zweierlei abdecken: Zum Einen das „Aufnehmen“ einer neuen Anfrage in Abhängigkeit des Zählervariablenstands der beiden SPS-I/O-Karten und zum Zweiten die Zwischenpufferung, wenn eine zweite Anfrage eintrifft, noch bevor die erste abgearbeitet worden ist. Abbildung 10 zeigt den zugehörigen Zustandsgraphen. Leicht erkenntlich ist an diesem Beispiel bereits, warum Warteschlangen nicht in der straight-forward-Variante realisiert werden können: Hätte man statt zweier (innerhalb der Bearbeitungszeit lediglich einmalig) anfragender SPSen, viele (möglicherweise in kurzen Abständen) anfragende Komponenten, so wäre der Graph weitaus komplexer und die Zustandsexplosion nur noch durch eine weitere Abstraktionsstufe in der Darstellung zu verschleiern. Würde man Warteschlangen, wie sie z.B. an Switchen auftreten, direkt modellieren, würde dies zu einer Vergrößerung des Zustandsraumes führen, welche die Modelle schlichtweg nicht mehr analysierbar macht. Geeignete Alternativen zur Lösung dieses Problemes werden derzeit untersucht.

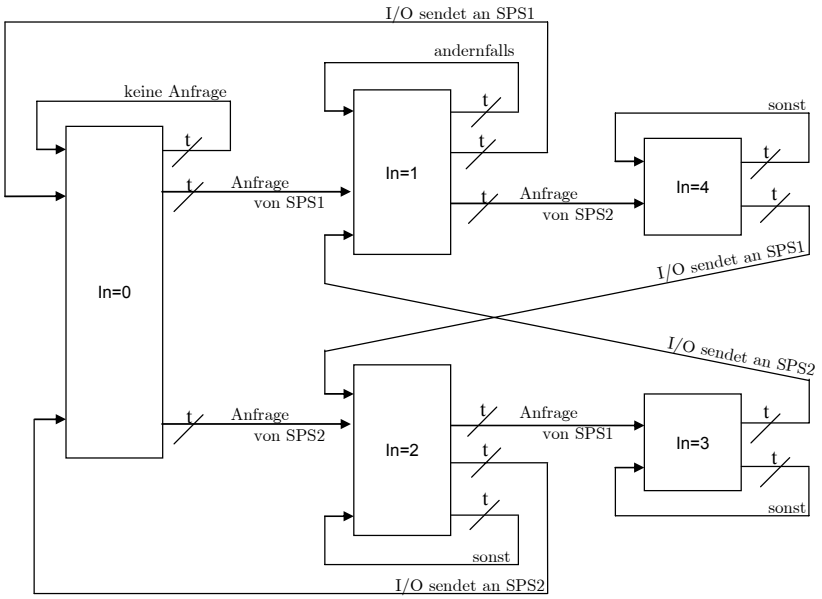

Abbildung 10: I/O-Board-Eingangs-Modul.

Die beiden SPS-IOs benötigen streng genommen ebenfalls einen Eingangspuffer. Dieser wurde in dieser Implementierung jedoch insofern vernachlässigt, als dass lediglich je ein Modul geschaffen wurde, dass feststellt, ob eine gesendete Anfrage beantwortet wurde, oder nicht. Wichtig in der Umsetzung ist noch, dass in der gewählten Implementierung nur fiktive Datenpakete bewegt werden, also keine Information darüber vorhanden ist, ob bereits eine Anfrage abgesetzt wurde. Dies abzufangen ist ebenfalls Aufgabe obiger Module. Der zugehörige Automatengraph ist in Abbildung 11 gezeigt.

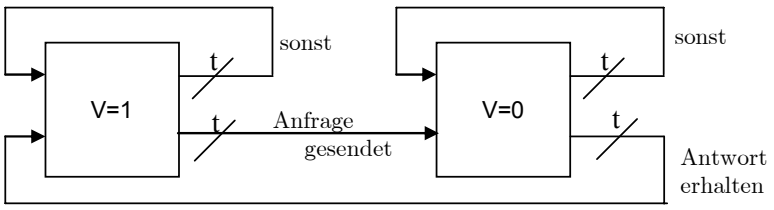

Abbildung 11: SPS-I/O-Antwort-erhalten-Modul.

Somit fehlen nur noch die beiden Signaltracking-Module. Das erste stellt das Erreichen der Abbruchbedingung fest, welche gegeben ist, wenn die maximale Signallaufzeit überschritten wird oder der Aktuator erfolgreich ausgelöst wurde. Das zweite „beobachtet“ insofern, als dass mitverfolgt wird, dass das Paket vom Sensor an der SPS-I/O-Karte angekommen ist, weitergeschaltet wird, sobald die SPS den Zykluspunkt „lesen“ das nächste Mal erreicht und wiederum ein weiterer Schritt registriert wird, wenn sich das Paket auf dem Weg zum Aktuator am fünften I/O-Board befindet. Da das Abfragen und die Wertzuweisung der I/O-Boards von der SPS-I/O im

gleichen Paket erfolgt, ist für diesen Vorgang keine weitere Modellierung notwendig. Sobald das fünfte I/O-Board die Bearbeitung abgeschlossen hat, gilt der Ausgang als gesetzt. In diesem Fall wird noch die Laufzeit wie in Abschnitt 2.4 erläutert übergeben und die Ausführung beendet.

3.2 Ergebnisse

Abbildung 12 zeigt die logarithmisch skalierte Wahrscheinlichkeit unterschiedlicher Reaktionszeiten. Leichte Schrägen im Graphen sind durch die verwendete Synchronisationsschrittweite entstanden. Der Wert von 7,24% bei 20ms bedeutet, dass die Wahrscheinlichkeit für eine Reaktionszeit zwischen 19,75ms und 20ms in absoluten Zahlen 1,81% beträgt. Bezieht man diesen Wert auf Synchronisationsschrittweite von 0,25 ergeben sich obige 7,24%, womit im Grenzübergang der Synchronisationsschrittweite gegen Null eine Fläche unter der Kurve von 1,0 gewährleistet ist, weshalb die Punkte in Abbildung 12 miteinander verbunden wurden. Der Wert von 1,81% wurde unter Prüfung der PCTL-Formel

P=? [true U Runtime=Lf]

auf das sich aus den Teilautomaten ergebende Gesamtsystem erhalten, wobei für Lf in diesem Fall 80 (20ms dividiert durch die Synchronisationsschrittweite von 0,25) eingegeben wurde, während Runtime, wie zuvor beschrieben, vom Modell gesetzt wurde. Die anderen Punkte sind in analoger Weise entstanden.

Die schnellste Reaktionszeit (9ms) entsteht dann, wenn die Anfrage von SPS_1 gerade zum Zeitpunkt des Signalwechsels eintrifft und sich keine weitere Verzögerung durch die SPS-IO einstellt (Summe aus 3 Switchübergängen (je 0,125ms), zwei Bearbeitungszeiten eines I/O-Boards (je 1,5ms) und dem Maximum aus Zykluszeit SPS und Zykluszeit SPS-I/O). Ab 12ms sind die Reaktionszeiten aufgetragen, bei denen SPS und SPS-I/O-Zyklus nicht mehr harmonieren. Die kleine Spitze zwischen 15,5 und 17ms ist durch die Bearbeitungszeit am I/O-Board einerseits und die Überlagerung der beiden SPS-IOs gegeneinander – wodurch Anfragen warten müssen – verursacht. Diese Überlagerung ist auch für das zweite Plateau verantwortlich. Das anschließende anderthalb millisekündige quadratische Absinken ist dem doppelten Auftreten der Bearbeitungszeit (I/O-Board 4 als Sensor und I/O-Board 5 als Aktuator) zuzuschreiben. Der hinterste Teil des Graphen vereinigt ungünstige Wartezeiten an den I/O-Boards mit nicht harmonisierenden Zyklen von SPS und SPS-I/O, woher die Dauer von 5ms rührt. Damit beträgt die worst-case-Verzögerung 26,75ms. Allerdings lässt sich eine Maximalverzögerung von 22ms mit einer 99,9%igen Sicherheit garantieren.

Greifeneder, J.; Frey, G.: *Analyse des Antwortverhaltens vernetzter Automatisierungssysteme mittels Probabilistic Model Checking.* Proceedings of the 9th Conference on Entwurf komplexer Automatisierungssysteme (EKA 2006), Braunschweig, Germany, pp.131-151, May 2006.

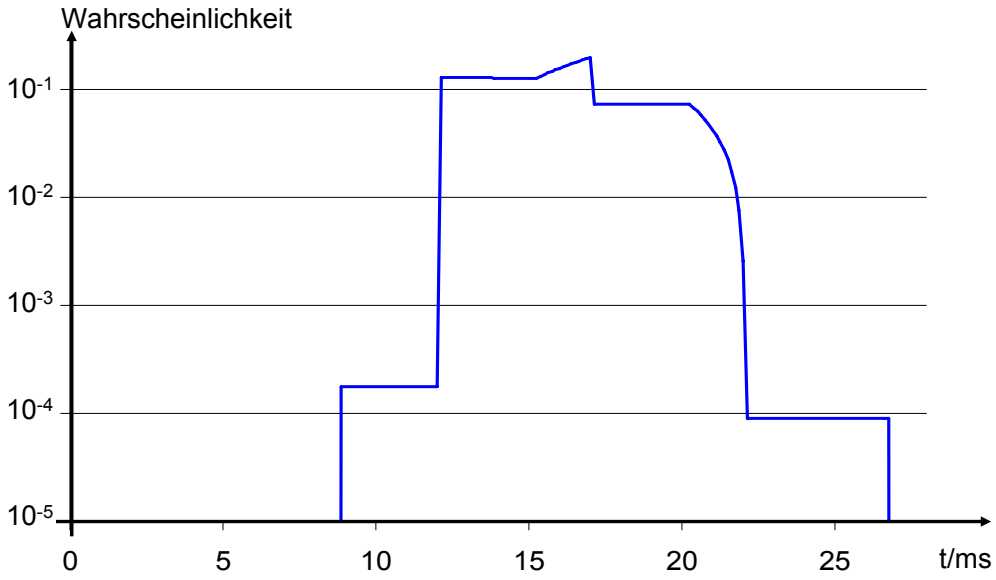

Abbildung 12: Wahrscheinlichkeit unterschiedlicher Reaktionszeiten in ms.

4 Zusammenfassung und Ausblick

Bei Verwendung von Probabilistic Model Checking (PMC) zur Analyse des Antwortverhaltens von netzwerkbasierten Automatisierungssystemen beeinflusst die Wahl der zu prüfenden Eigenschaften das zu formulierende Modell. Eine scharfe Trennung führt daher zu illusorisch großen Zustandsräumen und Verifikationszeiten. Daraus resultieren der vorgestellte Modellierungsansatz und die Notwendigkeit, diesem und der Beschreibung zu testender Eigenschaften besondere Aufmerksamkeit zu widmen. Anhand eines Anwendungsbeispieles wurde die Umsetzbarkeit des vorgestellten Ansatzes gezeigt. Dies illustriert, dass die Zuverlässigkeit von verteilten netzwerkbasierten AT-Systemen betreffenden Fragen unter Verwendung von PMC beantwortet und somit worst-case-Analysen vermieden werden können.

Die nächsten Meilensteine der vorgestellten Arbeit liegen in einer Verfeinerung des vorgestellten Anwendungsbeispieles sowie in der Diskussion von Möglichkeiten, Warteschlangen sinnvoll (d.h. ohne die zuvor geschilderte Zustandsexplosion) zu implementieren. Weiterhin wird die Ausarbeitung von Erfahrungsregistern zur Erstellung geeigneter Modul-Modelle mit dem Fernziel eines automatisierten computergestützten Designprozesses angestrebt.

Greifeneder, J.; Frey, G.: *Analyse des Antwortverhaltens vernetzter Automatisierungssysteme mittels Probabilistic Model Checking.* Proceedings of the 9th Conference on Entwurf komplexer Automatisierungssysteme (EKA 2006), Braunschweig, Germany, pp.131-151, May 2006.

Literatur

[AlCoDi1991] Alur, R., Courcoubetis, C., Dill, D: *Model-checking for probabilistic real-time systems.* ICALP'91, LNCS, **510**: 1--100, Springer, 1991.

[AlCoDi1993] Alur, R., Courcoubetis, C., Dill, D: *Model-Checking in Dense Real-time*. Inf. Comput. **104**(1): 2-34, 1993.

[BeBiFi2001] Bérard, B., Bidiot, M., Finkel, A. Laroussinie, F. Petit, A., Petrucci, L., Schnoebelen; Ph. (2001): *Systems and Software Verification, Model-Checking Techniques and Tools* Springer, 2001.

[DaAl2005] David, R., Alla, H.: *Discrete, Continuous, and Hybrid Petri Nets*. Springer, 2005.

[GreFr2005] Greifeneder, J., Frey, G.: *Probabilistic Delay Time Analysis in Networked Automation Systems.* Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2005, Catania, Italy, **1**, pp. 1065-1068, Sept. 2005.

[HanJo1999] Hansson, H., Jonsson, B.: *A logic for reasoning about time and reliability*, Formal Aspects of Computing, **6**, no. 4: 512--535, 1999.

[HenNi1992] Henzinger, T., Nicollin, X., Sifakis, J., Yovine, S.: *What good are digital clocks?* Proc. ICALP'92, LNCS*,* **623**: 545--558, Springer, 1992.

[KwNoPa2002] Kwiatkowska, M., Norman, G., Parker, D.: *PRISM: Probabilistic symbolic model checker.* Proc. TOOLS'02, LNCS, **2324**: 200-–204. Springer, 2002.

[KwNoPa2004] Kwiatkowska, M., Norman, G., Parker, D.: *Modelling and Verification of Probabilistic* Systems, in: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. *CRM Monograph Series*, **23**. American Mathematical Society, 2004.