Informatik aktuell

W. A. Halang
O. Spinczyk (Hrsg.)

Betriebssysteme und Echtzeit

Echtzeit 2015

Springer Vieweg

Informatik aktuell

Herausgegeben im Auftrag der Gesellschaft für Informatik (GI)

Wolfgang A. Halang
Olaf Spinczyk (Hrsg.)

Betriebssysteme und Echtzeit

Echtzeit 2015

Fachtagung des gemeinsamen Fachausschusses
Echtzeitsysteme von
Gesellschaft für Informatik e.V. (GI),
VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik (GMA) und
Informationstechnischer Gesellschaft im VDE (ITG)
sowie der Fachgruppe Betriebssysteme von GI und ITG
Boppard, 12. und 13. November 2015

GESELLSCHAFT FÜR INFORMATIK E.V.

VDI VDE VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik

ITG INFORMATIONSTECHNISCHE GESELLSCHAFT IM VDE

Springer Vieweg

CR Subject Classification (2001): C3, D.4.7

ISSN 1431-472X

ISBN 978-3-662-48610-8 e-ISBN 978-3-662-48611-5
DOI 10.1007/978-3-662-48611-5

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer-Verlag GmbH Berlin Heidelberg ist Teil der Fachverlagsgruppe
Springer Science+Business Media

www.springer-vieweg.de

Vorwort

Was ist ein Elefant? Die scherzhafte Antwort auf diese Frage, nämlich eine Maus mit einem Betriebssystem, ist ganz typisch für den Stand der Dinge und widerspricht völlig den Anforderungen, die der erste Unterzeichner vor 40 Jahren als Gasthörer in einer Vorlesung über Betriebssysteme an der Universität Dortmund, wo der zweite Unterzeichner heute tätig ist, gelernt und auch seither nicht mehr vergessen hat:

- effizient,
- verläßlich,
- unauffällig.

Insbesondere die letztgenannte Anforderung erfüllen heutige Betriebssysteme in keiner Weise. Hinzu kommt, daß um den Massenmarkt der im Büro- und Privatbereich eingesetzen Betriebssysteme hart gekämpft wird – ein Kampf, der oft Züge eines Glaubenskrieges annimmt. Um durch Bereitstellung von immer mehr Funktionalitäten neue Versionen verkaufen zu können, werden Betriebssysteme immer voluminöser und büßen so an Effizienz und Verläßlichkeit ein.

Gerade diese beiden Eigenschaften sind jedoch für Echtzeitsysteme und oft sicherheitsgerichtete eingebettete Systeme entscheidend. Weil sie auf denselben Hardware-Plattformen aufgebaut sind wie Büro- und Privatrechner, brauchen sie Programme, die zusammen mit den Eigenschaften von Digitalrechnern die Grundlagen ihrer möglichen Betriebsarten bilden und insbesondere die Programmabarbeitung steuern und überwachen, was genau der Definition von Betriebssystemen nach der Norm DIN 44300 entspricht.

Es sind also Betriebssysteme, die aus Digitalrechnern erst Echtzeitsysteme machen, indem sie die internen Abläufe in einer Betriebsart steuern, die die fundamentale Anforderung Rechtzeitigkeit erfüllt und dabei auch Effizienz und Verläßlichkeit gewährleistet.

Aufgrund dieser Bedeutung hat die Fachtagung Echtzeit in ihrer langen Geschichte schon mehrere Male das Hauptaugenmerk auf Echtzeitbetriebssysteme gelegt. Neu in diesem Jahr ist, dieses im Rahmen einer gemeinsamen Veranstaltung zusammen mit der Fachgruppe Betriebssysteme der Gesellschaft für Informatik und der Informationstechnischen Gesellschaft zu tun, die nach langem Vorlauf nun endlich zustandegekommen ist.

Der aktuelle Grund, das Thema Echtzeitbetriebssysteme erneut aufzugreifen, ergibt sich durch die stark wachsende Bedeutung der Vernetzung eingebetteter Systeme, die sich auch an Trends wie Cyber-Physical Systems oder Industrie 4.0 festmachen läßt. Diese Entwicklungen führen zu steigender Komplexität von Echtzeitsystemen und ihrer Betriebssysteme sowie zu der Frage, wie sich Echtzeitfähigkeit und Verläßlichkeit auch unter den Bedingungen oft nichtdeterministischen, fehleranfälligen und angreifbaren Netzverkehrs gewährleisten lassen und wie Echtzeitbetriebssysteme dabei trotzdem weiterhin kompakt bleiben können.

Im Rahmen der Steuerung industrieller Abläufe und von Verbrennungsmotoren widmen sich die ersten Beiträge der systematischen Validierung der Echtzeiteigenschaften mit verteilten Zustandsautomaten formulierter Software und deren Genauigkeit sowie einer Umgebung, die es ermöglicht, insbesondere im Hinblick auf den zukünftigen Einsatz von Mehrkernprozessoren die Reaktivität von und die Interaktion zwischen einer Fülle von Software-Modulen zu untersuchen, wie sie für Steuergeräte in der Kraftfahrzeugtechnik heute typisch sind.

Weil man Verläßlichkeit besser konstruktiv in der Architektur verankert, statt sie hinterher nachzuweisen, werden eine Prozessorarchitektur für sicherheitsgerichtete Echtzeitsysteme, die alle internen Ausfallmöglichkeiten erkennt und weitgehend toleriert, eine zeit- und rechenintensive Kognitions- mit Echtzeitregelungsmodulen integrierende Architektur für fernüberwachbare kooperierende Roboter sowie ein Ansatz vorgestellt, der die Replikation virtueller Maschinen zur hochverfügbaren Programmabsicherung dadurch für den Echtzeiteinsatz nutzbar macht, daß die Replikationszeitpunkte explizit vorgegeben werden können.

Es werden ein gestuftes Zuteilungsverfahren für Anwendungen mit weichen Echtzeitanforderungen sowie a priori unbekanntem und dann schwankenden Ressourcenbedarf, das Konzept abgeschirmter Abschnitte, das wartefreie Synchronisation von Echtzeitprozessen und damit latenzminimierte Betriebssystemkerne ermöglicht, sowie ein Betriebssystem entworfen, das bspw. in Automobilen durch konstruktive Fehlervermeidung und -toleranz als robuste Ausführungsumgebung auf unzuverlässiger Hardware dienen soll.

Mit verteilten Echtzeitsystemen beschäftigen sich Beiträge über Segmentierung parallelisierbarer Berechnungen nach einem Fork-Join-Prinzip, um sie dann auf Mehrkernprozessoren mit einfach determinierbarem Verhalten auszuführen, über automatisierte Analyse aus ereignisorientierter Sicht entworfener Echtzeitsysteme durch Abbildung auf zeitgesteuerte verteilte Systeme und über Erweiterung des AUTOSAR-Betriebssystems um ein Konzept zur kollaborativen, problematische Zustände vermeidenden Verwaltung der in Mehrkernsystemen den einzelnen Tasks zugeordneten Ressourcen.

Schließlich sind die preisgekrönten studentischen Abschlußarbeiten der Abschätzung des maximalen Energieverbrauchs eingebetteter Systeme mit Hilfe impliziter Pfadaufzählung oder genetischer Algorithmen, einer Testsuite zur Überprüfung von PEARL-Sprachsystemen auf Normenkonformität insbesondere hinsichtlich Operatoren, Synchronisation und Tasking sowie der bei Echtzeitsystemen wegen ihres vorhersehbaren Verhaltens möglichen gemeinsamen Analyse des Kontrollflusses von Anwenderprogrammen und Betriebssystemkern gewidmet.

Frau Dipl.-Ing. Jutta Düring gebührt unser herzlicher Dank dafür, daß sie zum wiederholten Male die Einreichungen redigiert und den vorliegenden Band konsistent und ansprechend gestaltet hat.

Hagen und Dortmund, im August 2015

Wolfgang A. Halang
Olaf Spinczyk

Inhaltsverzeichnis

Testen von Echtzeitsystemen

Testen von Echtzeiteigenschaften für verteilte Ablaufsteuerungen 1
Matthias Jurisch, Kai Beckmann

EMSBench: Benchmark und Testumgebung für reaktive Systeme 11
Florian Kluge, Theo Ungerer

Systemarchitekturen

Prozessorarchitektur zum Einsatz unter sicherheitsgerichteten Echtzeitbedingungen 21
Daniel Koß

SAKKORO. Eine generische, echtzeitfähige Systemarchitektur für kognitive, kooperierende Roboter 31
Adrian Leu, Danijela Ristić-Durrant, Axel Gräser

CPS-Remus: Eine Hochverfügbarkeitslösung für virtualisierte cyber-physische Anwendungen 39
Boguslaw Jablkowski, Olaf Spinczyk

Entwurfsaspekte

Ein hierarchisches Scheduling-Modell für unbekannte Anwendungen mit schwankenden Ressourcenanforderungen 49
Vladimir Nikolov, Franz J. Hauck, Lutz Schubert

Wartefreie Synchronisation von Echtzeitprozessen mittels abgeschirmter Abschnitte 59
Gabor Drescher, Wolfgang Schröder-Preikschat

*d*OSEK: Maßgeschneiderte Zuverlässigkeit 69
Martin Hoffmann, Florian Lukas, Christian Dietrich, Daniel Lohmann

Verteilte Systeme

Ein Fork-Join-Parallelismus in einer gemischt-kritischen Mehrprozessorumgebung 79
Marc Bommert

React in Time: Ereignisbasierter Entwurf zeitgesteuerter verteilter Systeme 89
Florian Franzmann, Tobias Klaus, Fabian Scheler, Wolfgang Schröder-Preikschat, Peter Ulbrich

Collaborative Resource Management for Multi-Core AUTOSAR OS 99
Renata Martins Gomes, Fabian Mauroner, Marcel Baunach

Graduiertenwettbewerb

Energieverbrauchsanalyse mittels impliziter Pfadaufzählung und genetischer Algorithmen 109
Peter Wägemann

Testanwendungen zur Überprüfung des PEARL-Sprachsystems auf Sprachkonformität 119
Christian Ritzler

Globale Kontrollflussanalyse von eingebetteten Echtzeitsystemen 128
Christian Dietrich, Martin Hoffmann, Daniel Lohmann

Testen von Echtzeiteigenschaften für verteilte Ablaufsteuerungen

Matthias Jurisch und Kai Beckmann

Hochschule RheinMain
Labor für Verteilte Systeme
65195 Wiesbaden
{matthias.jurisch|kai.beckmann}@hs-rm.de

Zusammenfassung. Im Bereich der industriellen Automatisierung hat die Software-Qualität und damit das systematische Testen eine hohe Bedeutung. In dieser Arbeit wird die Erweiterung eines Testframeworks um die Validierung von Echtzeiteigenschaften für verteilte Zustandsmaschinen vorgestellt. Die Arbeit wurde zusammen mit einem industriellen mittelständischen Kooperationspartner im Bereich industrieller Ablaufsteuerungen durchgeführt. Der Aufbau der domänenspezifischen Testmodellierung und die Testauswertung der Echtzeiteigenschaften als Teil eines modellgetriebenen Testprozesses werden beschrieben, und als Konsequenz aus der Synchronisationsungenauigkeit der Uhren des verteilten zu testenden Systems wird eine Fehlerabschätzung gegeben, die bei der Testauswertung berücksichtigt wird.

1 Einleitung

Die Software-Qualität hat immer größeren Einfluss auf die Wettbewerbsfähigkeit von Unternehmen. Gerade in der Automatisierung können mangelnde Qualität und Fehler in Produkten Schäden anrichten und unnötige Kosten verursachen. Moderne Methoden der Software-Technik, wie das modellbasierte Testen [1], versprechen Produktivitätssteigerungen bei gleichzeitig verbesserter Qualität. Allerdings haben kleine und mittelständische Unternehmen (KMU) der Automatisierungsbranche oft Probleme, ihre existierenden Entwicklungsprozesse während des Alltagsgeschäfts zu modernisieren. Neben Kosten für neue Werkzeuge und Schulungen erlaubt das Tagesgeschäft selten den Aufwand, existierende Projekte an die neuen Prozesse anzupassen. Dementsprechend besteht Bedarf nach kostengünstigen, an die individuelle Problemdomäne anpassbaren Lösungen. Dabei ist die Modellierung und Validierung von Echtzeiteigenschaften von großer Bedeutung.

Dieser Beitrag stellt die Erweiterung einer Testspezifikationssprache für verteilte Zustandsmaschinen um Echtzeitaspekte vor, die als Teil eines Testframeworks im Rahmen eines Kooperationsprojekts mit einem mittelständischen Unternehmen der Automatisierungsbranche entstanden ist [2]. In dieser Branche sind Anwendungen häufig in Form von verteilten Zustandsautomaten als dezentrale Ablaufsteuerung umgesetzt. Der Kooperationspartner verwendet für die

W. A. Halang, O. Spinczyk (Hrsg.), *Betriebssysteme und Echtzeit,* Informatik aktuell,
DOI 10.1007/978-3-662-48611-5_1,

Entwicklung solcher Anwendungen ein eigenes Werkzeug, mit dem Automaten tabellarisch spezifiziert und für eine große Anzahl an Zielplattformen Programmcodegerüste generiert werden kann. Die Umsetzung der Anwendungsfunktionalität, insbesondere die Kommunikation und Synchronisation der verteilten Automaten, erfolgt manuell und ist damit fehleranfällig. Dies erfordert umfassende Tests, insbesondere auch Regressionstest nach Änderungen. Im Rahmen dieses Projekts wurde ein Testframework entwickelt, das es ermöglicht, modellgetrieben Tests für Systeme aus verteilten Automateninstanzen zu spezifizieren, zu verwalten, wiederholt automatisiert auszuführen und deren Ergebnisse zu dokumentieren. Die Testspezifikation erfolgt über eine an die Bedürfnisse der Tester angepasste Domain Specific Language (DSL), die auch Zugriff auf die existierenden Automatenmodelle bietet. Die Vorarbeiten für diesen modellgetriebene Testprozess und das Testframework werden in [3] beschrieben.

Für das Testen von Ablaufsteuerungen spielen Echtzeiteigenschaften eine signifikante Rolle. Die grundsätzliche Architektur des Testframeworks, auf dem der hier vorgestellte Ansatz aufbaut, wird zusammen mit der Problemstellung in Abschnitt 2 vorgestellt. Die Erweiterung dieses Testframeworks umfasst die Testmodellierung von Echtzeitaspekten auf Basis von sogenannten abstrakten Ereignissen, die ausgehend vom Systemmodell auftreten können. Über diesen abstrakten Ereignissen können zeitliche Ausdrücke als Constraints, wie beispielsweise Deadlines, Dauern oder Jitter spezifiziert werden, was in Abschnitt 3 erläutert wird. Für die Auswertung der spezifizierten Echtzeitanforderungen werden die real aufgetretenen Ereignisse mit den lokalen Zeitstempeln der verteilten Ablaufsteuerungen als Trace gegen das modellierte Sollverhalten validiert. Die verteilte Zeitbasis erschwert prinzipiell die Auswertung von Testaussagen und dementsprechend auch die Validierung von Echtzeitanforderungen [4]. Um diesen Umstand mit einzubeziehen, umfasst die hier beschriebene Testmodellierung auch die zeitliche Synchronisationsungenauigkeit der Zeitstempel der aufgetretenen Ereignisse. Die Fehlerbetrachtung wird vom Testorakel für die Auswertung der Testaussage berücksichtigt und in Abschnitt 4 dargestellt. Die konkrete Umsetzung im Rahmen des Testframeworks wird danach in Abschnitt 5 umrissen und der hier beschriebene Ansatz in Abschnitt 6 mit verwandten Ansätzen aus der Literatur verglichen. Eine Zusammenfassung mit einem Ausblick auf geplante zukünftige Anwendungsfelder wird abschließend in Abschnitt 7 gegeben.

2 Hintergrund und Problemstellung

Das zusammen mit einem industriellen Kooperationspartner entwickelte Testframework für verteilte Ablaufsteuerungen in der Automatisierung ermöglicht es, existierende Systemmodelle, die mit einem proprietären Werkzeug erstellt wurden, in einem modellgetriebenen Testprozess wiederzuverwenden. Die Grobarchitektur, die auch den Testprozess erkennen lässt, ist in Abbildung 1 dargestellt. Die DSL für die Testspezifikation wurde zusammen mit dem Projektpartner entwickelt. Während der Testmodellierung kann auf das importierte Systemmodell

zugegriffen und Strukturinformationen in der Testspezifikation referenziert werden. Dies bietet dem Testingenieur das ihm bekannte Umfeld.

Die Testschnittstelle zum System under Test (SUT) wird vom Projektpartner bereitgestellt und stellt für den Testprozess ausschließlich die durchgeführten Zustandsübergänge der verteilten Zustandsmaschinen als Ereignisse zur Verfügung. Diese Ereignisse referenzieren die jeweilige Zustandsmaschine mit Ausgangs- und Zielzustand und enthalten den lokalen Zeitstempel für den Zeitpunkt des Zustandsübergangs. Im Weiteren werden die einzelnen Ereignisse als Trace-Einträge und deren Sequenz, die die Ereignisse aller aktiven Zustandsmaschinen des SUT umfasst, als Trace bezeichnet. Der Trace wird für die Gewinnung der Testaussagen für sowohl funktionale als auch nicht-funktionale Tests verwendet. Es wird vorausgesetzt, dass die Uhren eine ausreichende Granularität besitzen um lokale Ereignisse kausal zu trennen und die Synchronisationsgenauigkeit der verteilten Uhren quantifizierbar ist.

Tests sind in Testsuiten organisiert, die aus einer Sequenz von Testfällen bestehen, welche nacheinander ausgeführt werden und in der Regel aufeinander aufbauen. Ein Testfall selbst besteht ebenfalls aus einer Sequenz von Testschritten, die den Test treiben oder eine Testaussage ermitteln können. Die verschiedenen Sequenzen spannen einen Baum auf, den die eigentliche Testausführung in Form einer Tiefensuche traversiert und die jeweiligen Aktionen ausführt. Beispielsweise wird für eine Gewinnung einer Testaussage ein Objekt instantiiert, welches das gewünschte Sollverhalten überprüfen kann und ihm der Trace zugeführt.

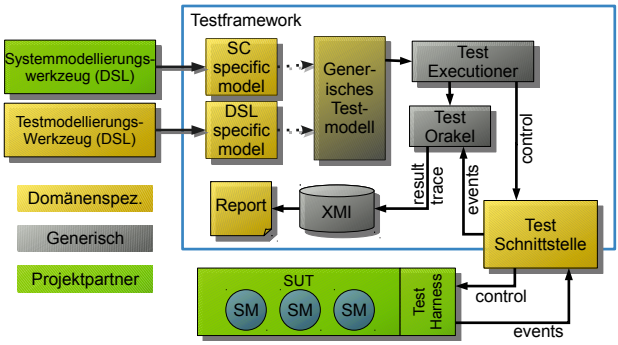

Abb. 1. Grobarchitektur des Testframeworks

Die Testmodellierung basiert auf Pfaden möglicher Zustandsübergänge der verteilten Zustandsmaschinen des SUT, die das Sollverhalten spezifizieren. Diese Pfade werden im Weiteren als globale Pfade bezeichnet und ein Beispiel ist in Listing 1 dargestellt. Die Formulierung drückt aus, dass nachdem der Automat *Schalter_1* den Zustand *Gedrueckt* betreten hat, der Automat *Tuersteuerung_1* die Zustände *Schliessend* und *Geschlossen* betreten muss. Mit diesen Pfaden wird das für die Gewinnung der Testaussage relevante Sollverhalten spezifiziert.

Die Auswertung beginnt, mit dem ersten Trace-Eintrag, der dem Anfang des Pfades zugeordnet werden kann.

Listing 1. Beispiel für einen globalen Pfad [5]

```
1 PathCondition Tuer_geht_zu {
2         Schalter_1:Gedrueckt -> Tuersteuerung_1:Schliessend
3         -> Tuersteuerung_1:Geschlossen
4 }
```

Damit das Testframework auch Echtzeiteigenschaften validieren kann, bedurfte es einer Erweiterung der Testmodellierung und -auswertung. Die Synchronisationsgenauigkeiten der verteilten Uhren sind dabei ein zu berücksichtigendes Problem. Zusammen mit dem industriellen Projektpartner wurden als zu testende Echtzeiteigenschaften Deadlines, Dauern zwischen möglichen abstrakten Ereignissen und verschiedene Jitter festgelegt.

3 Zeitliche Ausdrücke

Basis für die Modellierung und Prüfung von Echtzeiteigenschaften sind abstrakte Events, welche eine abstrahierte Darstellungsform für spezifizierbare Ereignisse in der Ausführung von verteilten Automaten repräsentieren. Diese Abstraktionsebene wurde gewählt, um verschiedene Arten von komplex und einfach zu formulierenden Ereignissen auf einer Ebene betrachten zu können. Über diesen Ereignissen lassen sich Einschränkungen von Dauern und Deadlines formulieren, deren zeitliche Werte mit Angabe einer zugehörigen Uhr definiert werden können. Diesen Einschränkungen wird ein Auswertungsintervall und ein Quantor zugeordnet, der es erlaubt, festzulegen, ob die Einschränkung immer, genau einmal oder mindestens einmal im Auswertungsintervall erfüllt sein muss. Ein einfaches Beispiel für eine formulierbare Einschränkung ist in Listing 2 dargestellt. Diese besagt, dass der Automat `A` spätestens, wenn auf der Uhr `Clock1` der Wert von 600 Millisekunden überschritten wird, den Zustand `A1` betreten muss. Die Auswertung soll beginnen, sobald die Uhr `Clock1` gestartet ist und enden, sobald diese den Wert von 5 Sekunden erreicht. Die an einer anderen Stelle der DSL definierbaren Uhren erlauben den Bezug auf den Beginn eines Testfalls oder einer Testsuite, aber auch auf die Systemzeit oder die Zeit, die vergangen ist, seitdem ein abstraktes Event das letzte mal aufgetreten ist. Die Synchronisationsgenauigkeit der Uhr kann ebenfalls angegeben werden. Jede Einschränkung besteht aus den folgenden Komponenten:

- Eine *Dauer* oder eine *Deadline-Spezifikation* (`enter B:B1`)
- Ein zeitlicher Minimal- und Maximalwert (`max 600 ms Clock1`)
- Der Anfang und das Ende eines Auswertungsintervalls (`start 0 s Clock1` und `stop 5 s Clock 1`)
- Der *Typ der Einschränkung* (`Eventually`)

Listing 2. Deadline für Betreten eines Zustands

```
1 Eventually {
```

```
2         enter A:A1
3         max 600 ms Clock1
4         start 0 s Clock1
5         stop 5 s Clock1
6 }
```

Die möglichen Bestandteile von Dauer- und Deadlinespezifikationen werden im Abschnitt 3.1 diskutiert. Zeitliche Angaben für Minimal- und Maximalwerte sowie für Anfang und Ende des Auswertungsintervalls sind relativ einfach und werden nicht weiter betrachtet. Die Einschränkungstypen werden in Abschnitt 3.2 vorgestellt.

3.1 Dauer- und Deadlinespezifikation

Tabelle 1. Abstrakte Events

Typ	Beispielhafte Formulierung
Betreten eines Zustands	`enter B:B1`
Verlassen eines Zustands	`leave B:X`
Auftreten einer Transition	`from A:A1 to A:A2`
Wahrwerden eines Prädikats	`turn true StatePredicate A1B2 { (A:A1 and B:B1) }`
Falschwerden eines Prädikats	`turn false StatePredicate Anot1Bnot2 { (not (A:A1)) and (not (B:B2)) }`
Beenden eines Pfadausdrucks	`end PathAssert fromA1toB2 { A:A1 -> * -> B:B2 }`

Deadline-Spezifikationen werden durch ein abstraktes Event formuliert. Für *Dauern* sind zwei mit `from` und `to` markierte abstrakte Events nötig, die den Anfang und das Ende der Dauer bestimmen. Tabelle 1 stellt die formulierbaren abstrakten Events anhand von Beispielen ihrer DSL-Darstellung vor. Die ersten drei Einträge beziehen sich direkt auf Zustandsübergänge. Diese Formulierungsmöglichkeit wurde ausgewählt, um auf der Abstraktionsebene, die auch bei der Systemmodellierung verwendet wurde, Einschränkungen ausdrücken zu können.

Ein Beispiel für die Zustandsübergänge eines verteilten Automatensystems und dem daraus entstehenden Trace ist in Abbildung 2 wiedergegeben. Der erste Trace-Eintrag (`X->B1`) passt zur beispielhaften Spezifikation des abstrakten Events *Betreten eines Zustands*. Auf ähnliche Weise können Trace-Einträge den abstrakten Events *Verlassen eines Zustands* und *Auftreten einer Transition* zugeordnet werden.

Um Aussagen über verteilte Zustände des Systems formulieren zu können, existieren die Fromulierungsmöglichkeiten *Wahrwerden* und *Falschwerden eines Prädikats*. Diese Prädikate werden als Aussagenlogische Formeln mit Zustän-

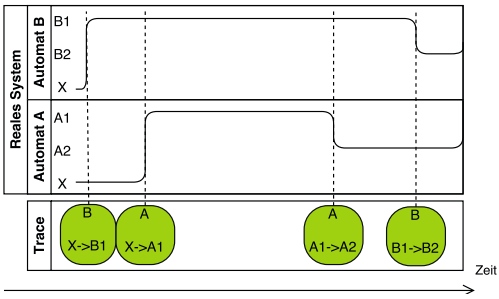

Abb. 2. Beispielhafter Trace [6]

den der jeweiligen Automaten als Aussagensymbolen formuliert. Sollen Trace-Einträge diesen Ausdrücke zugeordnet werden, müssen mehrere Trace-Einträge betrachtet werden. Das zweite in Abbildung 2 dargestellte Trace-Event (`X->A1`) passt zum in Tabelle 1 dargestellte Beispiel für das Wahrwerden eines Prädikats. Dieser Trace-Eintrag führt dazu, dass das Prädikat wahr wird. Die Prädikate werden also über verteilten Zuständen des Gesamtsystems ausgewertet.

Das *Beenden eines Pfadausdrucks* dient der Spezifikation des Zeitpunkts, an dem ein bestimmter Pfad abgearbeitet ist. So können Ausdrücke aus früheren Versionen der DSL wiederverwendet und zusätzlich mit zeitlichen Einschränkungen versehen werden. Auch hier muss eine Folge von Trace-Einträgen ausgewertet werden, die zu dem formulierten Pfad passen muss. Der erste Eintrag, der zur Erfüllung des Pfades führt, passt zu dieser Spezifikation. Der letzte Eintrag (`B1->B2`) des in Abbildung 2 dargestellten Trace passt zu der in Tabelle 1 vorgestellten Spezifikation für das Beenden eines Pfads.

3.2 Einschränkungstypen

Über die erlaubten *Typen der Einschränkungen* und deren Anwendbarkeit gibt die Tabelle 2 eine Übersicht. Die ersten drei Einträge sind Quantoren, die an ähnliche Ausdrucksweisen aus temporalen Logiken angelehnt sind. Des Weiteren spielt Jitter insbesondere bei der Ansteuerung von Maschinen in der Automatisierungstechnik eine wichtige Rolle und Summen von Dauern dienen zur Beurteilung einer Gesamtverweildauer in Zuständen. Werden Deadlines betrachtet, bezieht sich die Einschränkung auf den Zeitpunkt des abstrakten Events. Ist der Ausdruckstyp `Eventually` muss dieser Wert mindestens einmal im Auswertungsintervall innerhalb der über `min` und `max` festgelegten Grenzen liegen, bei `ExactlyOnce` genau ein mal und bei `Always` immer.

Bei Dauern beziehen sich die Einschränkung auf den zeitlichen Abstand zwischen den abstrakten Events der Dauern. Für die ersten drei Einträge der Tabelle ist die Bedeutung äquivalent zur Betrachtung von Deadlines. `JitterToAverage` betrachtet alle im Auswertungsintervall spezifizierten Dauern. Eingeschränkt wird dabei die maximale Abweichung zum Mittelwert aller Dauern im Intervall. Die Abweichung muss zwischen den mit `min` und `max` festgelegten Werten

Tabelle 2. Ausdruckstypen

Typ	Beschreibung	Deadlines	Dauern
`Eventually`	Mindestens ein Zutreffen im Intervall	✓	✓
`ExactlyOnce`	Genau ein Zutreffen im Intervall	✓	✓
`Always`	Aussage ist im Intervall immer korrekt	✓	✓
`JitterToAverage`	Abweichung einer Dauer zum Mittelwert immer innerhalb der Spezifikation	×	✓
`CycleToCycleJitter`	Abweichung zwischen aufeinanderfolgenden Dauern immer innerhalb der Spezifikation	×	✓
`SumOfDurations`	Summe der Dauern in einem Intervall innerhalb der Spezifikation	×	✓

liegen, `CycleToCycleJitter` legt die Abweichung zwischen aufeinander folgenden Dauern fest und bei `SumOfDurations` wird die Summe aller Dauern im Auswertungsintervall beschränkt.

4 Fehlerbetrachtung

Bei der Auswertung der Ausdrücke auf dem Trace muss beachtet werden, dass die Zeitstempel durch verteilte Uhren erzeugt werden und die auf den Uhren beobachteten Werte für ein Ereignis aufgrund der Synchronisationsungenauigkeit von einander abweichen können. Dem entsprechend ist eine Fehlerbetrachtung notwendig. Dabei wird angenommen, dass die Uhren des SUT mit einer maximalen Abweichung von g synchronisiert sind.

Werden Deadlines betrachtet, bezieht sich der gemessene Wert immer auf einen einzigen Trace-Eintrag. Dieser wird auf Basis einer Uhr bestimmt, die nach Voraussetzung mit dem maximalen Fehler g misst. Der maximale Fehler für Deadlines als Zeitpunkte ist daher $\Delta_{dea} = g$.

Dauern werden auf Basis von zwei Trace-Einträgen berechnet, die den Anfang und das Ende der Dauer bestimmen. Jeder dieser Trace-Einträge hat einen maximalen Fehler von g und das führt zu einem maximalen Fehler für Dauern von $\Delta_{dur} = 2g$.

Für den Jitter zwischen aufeinanderfolgenden Dauern spielen die Fehler der Dauern eine Rolle. Diese sind, wie gerade beschrieben, jeweils $\Delta_{dur} = 2g$. Die Abweichung zwischen den Dauern hat damit einen maximalen Fehler von $4g$. Der Fehler des Mittelwerts von Dauern kann nicht größer als $\Delta_{dur} = 2g$ sein. Da der Fehler für eine einzelne Dauer ebenfalls maximal $2g$ ist, ist der maximale Fehler für Jitter zwischen Mittelwert und einzelner Dauer ebenfalls $4g$. Der maximale Fehler für alle Arten von Jitter ist also $\Delta_{jit} = 4g$.

Da bei der Summe von Dauern der Fehler der Dauern addiert wird, übersteigt hier der Fehler bei n Dauern $n * 2 * g$ nicht. Diese Fehlerschranke ist allerdings in mehrerer Hinsicht nicht zielführend, um Tests sinnvoll durchführen zu können. Ein linear ansteigender Fehler mit der Anzahl der Messwerte steigt zu schnell, um Summen von vielen Dauern sinnvoll beurteilen zu können. Außerdem ist die