Vorab-Veröffentlichung, erscheint in: 9. Paderborner Workshop Entwurf mechatronischer Systeme, Heinz Nixdorf Institut, Paderborn, 2013.

Layout und Seitenzahlen können von der gedruckten Version abweichen!

Bewertung der Zuverlässigkeit selbstoptimierender Systeme mit dem LARES-Framework

Tobias Meyer, Christoph Sondermann-Wölke, Walter Sextro
Universität Paderborn
Pohlweg 47-49, 33102 Paderborn
Tel. 05251/60-1802, Fax. 05251/60-1803
E-Mail: {tobias.meyer|chrisson|walter.sextro}@uni-paderborn.de

Martin Riedl, Alexander Gouberman, Markus Siegle
Universität der Bundeswehr München
Werner-Heisenberg-Weg 39, 85579 Neubiberg
Tel. 089/6004-2417, Fax. 089/6004-2268
E-Mail: {martin.riedl|alexander.gouberman|markus.siegle}@unibw.de

Zusammenfassung

Selbstoptimierende mechatronische Systeme bieten die Möglichkeit, ihr Verhalten an geänderte Umgebungsbedingungen anzupassen. Dazu werden beispielsweise redundante Strukturen genutzt, Reglerparameter angepasst oder Regelstrategien umgeschaltet. Dies kann auch genutzt werden, um die Zuverlässigkeit des Systems zu steigern. Zugleich entstehen aber durch die gesteigerte Komplexität dieser Systeme zusätzliche Risiken. Um sicherzustellen, dass das System dennoch die gestellten Anforderungen bezüglich der Zuverlässigkeit erfüllt, ist eine Modellierung des Gesamtsystems und anschließende Zuverlässigkeitsbewertung notwendig. Dies ist aufgrund der situationsabhängigen Verhaltensanpassung und des nicht intuitiv vorhersehbaren Verhaltens jedoch nicht mit klassischen Verfahren möglich. Ein Modellierungsverfahren, das diese Eigenschaften abbilden kann, ist LARES (LAnguage for REconfigurable dependable Systems).

Die Anwendung von LARES zur Bewertung der Zuverlässigkeit eines selbstoptimierenden Systems wird anhand des Feder-Neige-Moduls gezeigt. Es ist eine Baugruppe der Fahrzeuge eines innovativen Bahnsystems, der RailCabs. Das Feder-Neige-Modul dient dazu, unerwünschte Schwingungen des Fahrzeugaufbaus zu minimieren. Mit LARES können die Hardware-Komponenten des Systems, ihre in Abhängigkeit von der aktuellen Situation veränderten Belastungen sowie die nicht-deterministische Verhaltensadaption modelliert werden.

Schlüsselwörter

LARES, Zuverlässigkeit, Verlässlichkeit, Selbstoptimierung.

1 Einleitung

Dank der zunehmenden Verfügbarkeit leistungsfähiger, für den Einsatz in Produkten des Maschinenbaus geeigneter Computer ist es möglich, intelligente Maschinen zu entwickeln, die ihr Verhalten selbstständig anpassen. Eine Möglichkeit, eine solche Intelligenz umzusetzen, stellen selbstoptimierende Systeme dar. Sie sind in der Lage, sich über Auswahl und Priorisierung verschiedener Ziele an geänderte Anforderungen der Nutzer und veränderte Umgebungsbedingungen zu adaptieren [GRS09b]. Dabei kann sich, durch die systemimmanente Reaktion auf äußere Einflüsse, ein für den Entwickler nicht im Vorfeld für die gesamte Nutzungsdauer absehbares Verhalten ergeben. Dabei kommt es allerdings, bedingt durch die Anpassung des Systemverhaltens an äußere Bedingungen und an den Systemzustand selbst, zu einer stark variierenden Belastung der einzelnen Systemkomponenten. Das Mehrstufige Verlässlichkeitskonzept bietet die Möglichkeit, trotz der resultierenden, nicht genau bekannten Belastungen der Komponenten, eine vorgegebene Nutzungsdauer zu erreichen [SS10, GRS09a]. Bei der Berechnung der Zuverlässigkeit des Gesamtsystems muss dies berücksichtigt werden.

Es ist daher notwendig, das System mit allen relevanten Systemkomponenten zu modellieren und die Verhaltensanpassung an den Systemzustand, die Umgebungsbedingungen und die Anforderungen der Nutzer einzubeziehen. Die Modellierungssprache LARES ist imstande, dies zu leisten.

Abschnitt 2 erläutert die Umsetzung der Verhaltensadaption selbstoptimierender Systeme und einige Ansätze zur Modellierung mechatronischer Systeme. Abschnitt 3 bietet eine Einführung in die Modellierungssprache LARES, bevor in Abschnitt 4 die Modellierung eines selbstoptimierenden Systems gezeigt wird; in Abschnitt 4.3 werden die erzielten Ergebnisse vorgestellt. Abschnitt 5 fasst die Erfahrungen kurz zusammen.

2 Verlässlichkeit selbstoptimierender Systeme

Das Mehrstufige Verlässlichkeitskonzept nutzt die Möglichkeit, das Systemverhalten an die aktuelle Degradation des Systems anzupassen. Es bestimmt dazu den aktuellen Systemzustand mit einer Zustandsüberwachung und klassifiziert diesen in vier Stufen. Es wird als Bestandteil der Informationsverarbeitung im Operator-Controller-Modul (OCM) eingebettet. Innerhalb des OCM, das die Informationsverarbeitung strukturiert [GRS09b], findet die Auswahl der aktuellen Ziele statt; zur Adaption des Verhaltens wird ein vorher berechneter optimaler Betriebspunkt ausgewählt, der über die Konfigurationssteuerung Änderungen in den genutzten Reglern umgesetzt wird.

Trotz der Implementierung des Mehrstufigen Verlässlichkeitskonzepts muss bei der Planung des Einsatzes des Systems oder der Wartungsphasen die im Vorfeld geschätzte Lebensdauer berücksichtigt werden. Dies geschieht über eine umfassende Bewertung der Verlässlichkeit des Systems, die insbesondere die Zuverlässigkeit berücksichtigt.

2.1 Stand der Technik

Zur quantitativen Zuverlässigkeitsbewertung technischer Systeme wird in kombinatorische und zustandsbehaftete Methoden unterschieden. Zu den kombinatorischen Methoden gehören die Fehlzustandsbaumanalyse [DIN61025] und die Zuverlässigkeitsblockdiagramme [Bir07]. Diese beiden Methoden bilden jedoch aufgrund ihres statischen Charakters, d.h. zeitliche und funktionale Abhängigkeiten können nicht modelliert werden, selbstoptimierende Systeme unzureichend genau ab. Zustandsbehaftete Methoden sind beispielweise Markov-Modelle [VDI4008] oder Petri Netze [MT95]. Der Einsatz von Markov-Modellen ist wünschenswert, da diese die auftretenden Abhängigkeiten der Komponentenausfälle abbilden können. Sie haben jedoch den Nachteil der Gefahr einer Zustandsraumexplosion für Modelle komplexer Systeme. Darüber hinaus ist das manuelle Aufstellen von Markov-Modellen fehleranfällig und zeitintensiv. Petri Netze bieten eine anschauliche Modellierung von Prozessabläufen, werden jedoch ebenfalls bei großen Systemen schnell unübersichtlich. Eine Variante, die Vorteile von Fehlzustandsbäumen und Markov-Modellen zu verknüpfen, stellen Dynamische Fehlzustandsbäume dar [DBB92]. Hierbei werden einzelne Gatter des Fehlzustandsbaums mit Markov-Modellen modelliert, um bestimmte dynamische Aspekte wie Standby-Redundanzen zu berücksichtigen. Wie bei statischen Fehlzustandsbäumen muss für jedes unerwünschte zu untersuchende Ereignis ein Fehlzustandsbaum aufgestellt werden. Die Modellierungssprache LARES bietet hingegen eine einfache Modellierungsoberfläche, welche durch eine automatische Transformation die Modellierungs- und Analysefähigkeiten von Markov-Modellen nutzt und so die Anforderung hinsichtlich der Zuverlässigkeitsbewertung selbstoptimierender Systeme erfüllt.

Es gibt einige zu LARES ähnliche Ansätze zur Modellierung fehlertoleranter verlässlicher Systeme. Zwei der maßgeblichen Vertreter, SLIM und AltaRica, werden im Folgenden beschrieben.

Auch in der Sprache SLIM (System-Level Integrated Modeling) [BCR+09] können Fehlerfortpflanzung, zufällige oder beständige Ausfälle bzw. leistungsreduzierter Betrieb abgebildet werden. Im Unterschied zu LARES ist SLIM vorwiegend für die Modellierung von Hardware/Software Systemen gedacht und an AADL (inkl. Error-Annex) angelehnt, um eine möglichst einfache Abbildung zu ermöglichen [BCC+10]. SLIM bietet unter anderem lokale Variablen und sprachliche Mittel zur Modellierung von Echtzeit- bzw. hybriden Systemen. Auch werden unterschiedliche Löser, sowohl zur Analyse der Leistungsfähigkeit und Zuverlässigkeit als auch zur Modellverifikation angebunden. Während mit SLIM mittels lokaler Zustände (Modes) bestimmte Systemeigenschaften über das Schalten sogenannter Mode-Transitionen abgebildet werden, gibt es in LARES zusätzlich ein Konzept zur Definition einer synchronen Reaktion über die verschiedenen Hierarchieebenen hinweg. Dann erhält man ein bestimmtes synchronisiertes Folgeverhalten in den einzelnen Verhaltensinstanzen, ohne lokale Zwischenzustände zu erzeugen.

Auch AltaRica [PR99] ermöglicht die direkte Modellierung hierarchischer diskreter Systeme. Allerdings wird AltaRica vorwiegend zur Modellverifikation funktionaler Aspekte genutzt. Eine Erweiterung zur Modellierung nicht-funktionaler Aspekte wird in [GPK+11] beschrieben. Allerdings umfasst auch dieser Ansatz nicht die innewohnende Stochastik verlässlicher Systeme, sondern beschränkt sich auf funktionale Abhängigkeiten, um die maßgeblichen Pfade in Fehlerzustände zu berechnen.

3 LARES: Modellierungs- und Analyse-Framework für fehlertolerante Systeme

LARES (LAnguage for REconfigurable dependable Systems) [WGR+09, RS12] ist eine Sprache zur Beschreibung der Struktur und des Verhaltens fehlertoleranter Systeme. Mit LARES lassen sich nicht nur einfache Redundanzstrukturen, sondern auch komplexe Fehlersituationen sowie aufwändige Reparatur- und Rekonfigurationsstrategien beschreiben.

Die Sprache beinhaltet Konstrukte zur Beschreibung stochastischer Aspekte, z.B. zur Definition von Lebensdauerverteilungen oder von Übergangswahrscheinlichkeiten zwischen verschiedenen Betriebszuständen. Für die quantitative Analyse eines LARES-Modells wird dieses in eine low-level Darstellung in Form eines Prozessalgebra-Modells oder direkt in eine Markovkette transformiert. Für die Modellerstellung und sämtliche Transformations- und Analyseschritte steht ein leistungsfähiges Framework zur Verfügung.

3.1 Grundlegende Sprachmittel von LARES

Ein LARES-Modell setzt sich aus den Konstrukten `Behavior` und `Module` zusammen. Ein `Behavior` beschreibt das lokale Verhalten einer Systemkomponente in Form eines Zustandsautomaten. Eine Transition zwischen zwei Zuständen kann nach einer exponentiell verteilten Zeit oder unmittelbar zeitlos schalten. Weiterhin kann für eine Transition ein sog. Guardlabel definiert werden, welches von außen (in Modulen) getriggert werden kann, um diese Transition zu aktivieren. Transitionen ohne Guardlabel sind automatisch aktiviert.

Ein `Module` stellt eine abstrakte Definition einer Komponente dar, die aus einer Teilmenge der `Behavior`- oder anderen `Module`-Definitionen erben und diese dadurch instanziieren kann. Die `System`-Definition ist eine spezielle `Module`-Definition, die zugleich auch die Wurzelinstanz des Systems darstellt. Rekursiv entsteht damit ein Instanzbaum mit instanziierten `Behavior`-Definitionen (im Folgenden kurz *Verhaltensinstanzen*) in den Blattknoten, der die Struktur des Modells beschreibt. Der Zustandsraum des Gesamtsystems setzt sich durch Komposition der Zustandsräume von Verhaltensinstanzen zusammen.

Das globale Verhaltensmodell wird durch Interaktionen in Form von Ereignissen beschrieben, die das lokale Verhalten der Verhaltensinstanzen steuern. Ein Ereignis wird dabei in einem Modul syntaktisch durch den Ausdruck `<Bedingung> guards <Reaktion>` definiert und kann eine Kaskade von weiteren Reaktionen durch die Instanzhierarchie nach sich ziehen. Hierbei gibt `<Bedingung>` einen logischen Ausdruck über Zustände der Verhaltensinstanzen an. Auf Grund der durch die Instanzhierarchie entstehenden Kapselung können Zustände in Form von `Condition`-Ausdrücken durch den Instanzbaum in Richtung Wurzel weitergereicht und durch logische Verknüpfungen mit anderen Bedingungen modifiziert werden. Analog muss auch die Reaktion eines Ereignisses durch den Instanzbaum zu den Verhaltensinstanzen hin propagiert werden. Hierzu können in den Modul-Definitionen sog. Forwardlabels in einem `forward <label> to`-Ausdruck definiert werden, der seinerseits ein Folgeereignis in der ausgelösten Kaskade beschreibt und weitere Forwardlabels oder schlußendlich Guardlabels in Verhaltensinstanzen aktivieren kann. Die einfachste Form einer Reaktion ist die Aktivierung eines einzigen Guardlabels bzw. Forwardlabels. Im Allgemeinen kann eine Reaktion, und damit der erreichte Folgezustand, auch weitere Ereignisse auslösen und/oder die dadurch resultierenden Transitionen synchronisieren. Durch die Menge von aktivierten Transitionen entstehen mögliche Zustandsübergänge, die zusammen eine diskrete Wahrscheinlichkeitsverteilung über die Menge der Nachfolgezustände induzieren.

3.2 Framework und Übersetzung in Stochastische Prozessalgebra

Das LARES-Framework basiert auf Eclipse und stellt eine integrierte Entwicklungsumgebung (IDE) für LARES dar, die sich aus einem Editor- und einem Analyse-Plugin zusammensetzt [GRS+12], welches die in der LARES-Bibliothek spezifizierten Transformationen und Analysen anbindet.

Der Editor unterstützt den Modellierer mittels Syntaxhervorhebung, Code-Vervollständigung und Syntax- sowie Referenzprüfung, um ein syntaktisch korrektes bzw. teilweise semantisch geprüftes Modell zu erstellen.

Das Analyse-Plugin erleichtert dem Nutzer die Übersicht und Auswertung des Modells. Aus der LARES-Spezifikation kann unter anderem der Instanzbaum und der Zustandsraum erzeugt und graphisch ausgegeben werden. Des Weiteren werden die in der Spezifikation eingefügten Maße extrahiert. Sie ermöglichen, z.B. durch den angebundenen Prozessalgebra-Löser CASPA [BRS+09], transiente oder stationäre Maße zu berechnen. Die Anbindung erfolgt mittels einer vollständig automatisierten Transformation der LARES-Modelle in eine Prozessalgebra-Spezifikation [RS12].

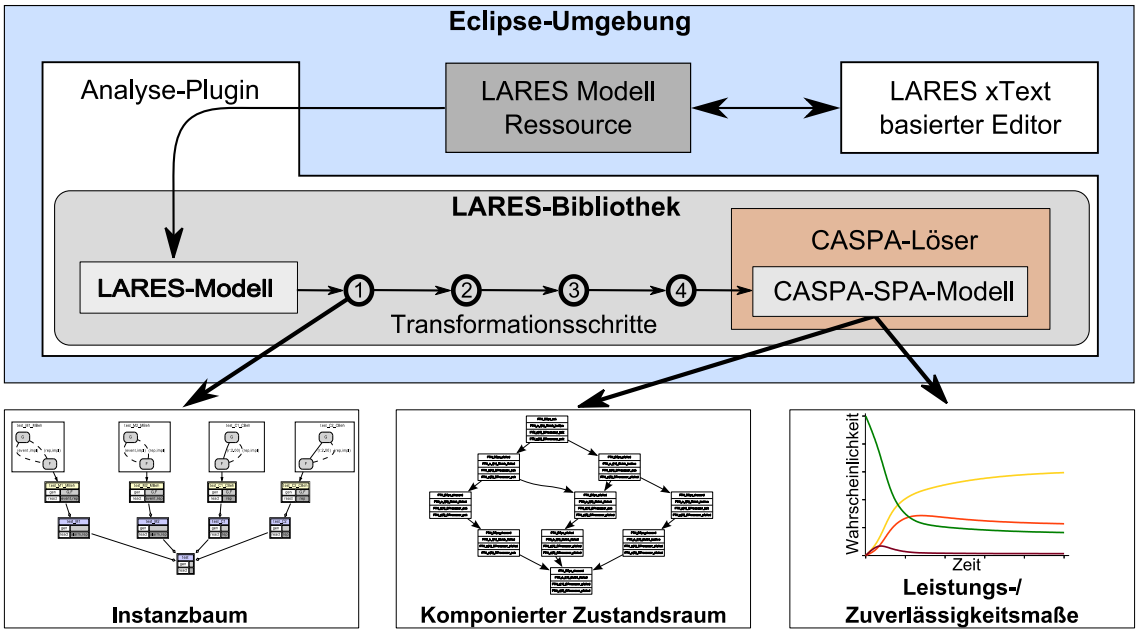

Bild 1: LARES-Entwicklungsumgebung

Wie in Bild 1 dargestellt, wird die Transformation vom LARES-Modell in das vom CASPA-Löser nutzbare CASPA-SPA-Modell in vier Teilschritten durchgeführt: aufgrund der Möglichkeit, Modul- und Verhaltensdefinitionen zu parametrieren, müssen Instanzen mit den für sie gültigen Parametern erzeugt werden (1). Dazu werden sukzessive bei der Instanziierung parametrierte Ausdrücke aufgelöst und gleichzeitig definierte Initialbelegungen für die erzeugten Subinstanziierungen gesetzt. Als nächster Schritt (2) werden alle `Conditions` bzw. generativen Ausdrücke aufgelöst. Dies heißt, dass Boole'sche Ausdrücke über aggregierte Zustandsinformation rekursiv bis auf atomare Einzelzustände der Verhaltensinstanzen im Modell verfeinert werden. Ein weiterer Schritt (3) verfährt ebenso mit den reaktiven Ausdrücken, wobei diese auf die Guardlabels der Verhaltensinstanzen abgebildet werden. In einem finalen Schritt (4) wird die Hierarchie aufgelöst. Dazu werden die Verhaltensinstanzen in sequentielle SPA Prozesse umgewandelt und entsprechend der Instanzstruktur und der vorgegebenen Synchronisationen im rekursiven Abbau komponiert. Im automatisierten Ablauf wird CASPA nun entweder für eine einfache Zustandsraumanalyse genutzt oder um die Maße entsprechend ihrer Definitionen, beispielsweise die Ausfallwahrscheinlichkeit, zu errechnen.

Die LARES-Bibliothek ermöglicht zudem eine direkte Abbildung auf Beschriftete Markovketten, die verwendet werden kann, um die SPA Transformation zu überprüfen. Jedes Modell einer Testreihe wird über ihre SPA Abbildung generiert und das Ergebnis mittels starker Bisimulation auf Verhaltensäquivalenz zu der jeweils direkt erzeugten Markovkette geprüft. Aufgrund der Diversität beider Transformationen ist nicht davon auszugehen, dass beide Ergebnisse bei inkonsistenter Definition der Semantik oder falscher Implementation zufällig verhaltensäquivalent sind (siehe auch [RS12], Sect. 6).

4 Modellierung Selbstoptimierender Systeme mit LARES

Im Folgenden wird anhand des Feder-Neige-Moduls gezeigt, wie die Möglichkeiten der Modellierungssprache LARES mit dem zugehörigen Framework genutzt werden können, um die Zuverlässigkeit eines selbstoptimierenden Systems zu bestimmen.

4.1 Beispielsystem: Feder-Neige-Modul

Bei der Neuen Bahntechnik Paderborn wurden neuartige Schienenfahrzeuge, genannt RailCabs, entwickelt, die autonom fahrend direkten Verkehr vom Start zum Ziel bieten. Sie wurden zur Entwicklung modularisiert, wobei ein Modul, das Feder-Neige-Modul, der Entkopplung des Aufbaus von den hochfrequenten Schienenanregungen dient. Es verfügt zu diesem Zwecke über zwei Aktorgruppen, die aus jeweils einem 3D-Positionssensor und drei Zylindermodulen bestehen. Diese bewegen den Fußpunkt einer glasfaserverstärkten Kunststofffeder, über die das gesamte Modul auf der zugehörigen Achse lastet, und sind somit in der Lage, zusätzliche Kräfte zum Dämpfen der Aufbaubewegungen aufzubringen. Die Zylindermodule bestehen aus je einem Hydraulikzylinder mit integriertem Wegaufnehmer und zugehörigem Signalverstärker sowie einem Ventil mit Ansteuerkarte. Sie bilden eine Redundanzstruktur, die mit Hilfe von Selbstoptimierung genutzt werden kann [MKS+13]. Zusätzlich verfügt das Feder-Neige-Modul über fünf Beschleunigungssensoren, die die Bewegung des Aufbaus detektieren. Diese dienen der den Zylindermodulen übergeordneten Regelung der Aufbaubewegung zur Bestimmung der Ist-Position.

4.2 Modellbildung

Die Hierarchie des realen Prüfstands kann durch den Einsatz der Modellierungssprache LARES auch im Modell erhalten bleiben. Dies führt zu einer sehr einfachen Abbildung des mechanischen und des elektrischen Teils des Systems. Zusätzlich muss die Informationsverarbeitung berücksichtigt werden, da sie über die Adaption des Verhaltens und die dadurch veränderte Nutzung einen maßgeblichen Einfluss auf die Ausfallrate der Aktoren hat. Um die externen Stimuli des Systems abzubilden, sind weiterhin Modelle der Umgebung und der vom Nutzer des Systems vorgegebenen Anforderungen notwendig.

4.2.1 Modellierung von Komponentenfehlern

Um den hierarchischen Aufbau des Systems aufrecht zu erhalten, wird für jede Systemkomponente ein separates `Module` erstellt. Geringen Modellierungsaufwand verursachen dabei die Beschleunigungssensoren: Als elektronische Komponenten können sie über die Zustände *functional* und *failed* sowie eine Ausfallrate beschrieben werden.

Die Zylindermodule hingegen, die nicht weiter in Subkomponenten[1] unterteilt werden, können auf zwei verschiedene Arten ausfallen. Einerseits ist dies das Abreißen eines Schlauchs, der Bruch einer Verbindung oder eine Leckage am Zylinder selbst, was dazu führt, dass der Zylinder keine Kräfte mehr aufbringen kann; der Fehlermode wird als *loose* bezeichnet. Geht andererseits ein Teil der Regelschleife, bestehend aus Wegaufnehmer, Signalverstärker, Ventil und Ansteuerkarte kaputt, sitzt der Zylinder in bekannter Position fest, er ist *stuck*, bringt aber nach wie vor Kräfte auf. In diesem Fall kann das System rekonfiguriert und weiterhin genutzt werden. Die Wahrscheinlichkeit dieses Fehlers ist von der aktuellen Belastung des Systems abhängig.

Bild 2 zeigt das gewählte Modell eines Zylindermoduls. Die beiden Ausfälle *stuck* und *loose* werden in separaten `Behavior` spezifiziert. Ein übergeordnetes `Module M_CylinderModule` fasst diese zusammen. Die hierzu notwendigen Transitionen sind von den Zuständen `B_Stuck.stuck` und `B_Loose.loose` abhängig.

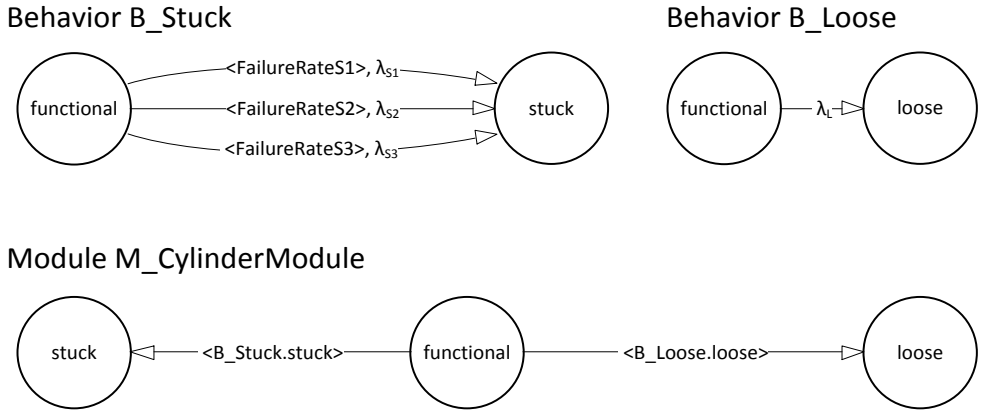

Bild 2: Modell[2] des Ausfallverhaltens eines Zylindermoduls für die Fehlerfälle loose und stuck (Leicht vereinfachte Darstellung). In Abhängigkeit der Bedingungen `<FailureRateS1…3>` wird die Ausfallrate λ_{S1} bis λ_{S3} zwischen den Zuständen `functional` und `stuck` aktiviert.

4.2.2 Modellierung lastabhängiger Ausfallraten

Die Modellierung lastabhängiger Ausfallraten teilt sich in zwei Bereiche auf: Einerseits muss die Verbindung vom aktuellen Systemverhalten zur jeweiligen Ausfallrate herge-

[1] Die Zylindermodule bestehen zwar, wie bereits in Abschnitt 4.1 erläutert, aus mehreren einzelnen Komponenten. Es lässt sich jedoch durch eine detailliertere Modellierung kein weiterer Aspekt des Ausfallverhaltens abbilden, da jede der Subkomponenten eines Moduls im Sinne einer Reihenschaltung notwendig ist.

[2] Als Kreis dargestellt sind die möglichen Zustände der `Behavior`- bzw `Module`-Definition; Pfeile hingegen stehen für Transitionen. Bei stochastischen Transitionen ist die Übergangsrate λ mit angegeben; bei abhängigen Transitionen ist die notwendige Bedingung in <…> angegeben. Dabei wird `= true` implizit mit angenommen, aber nicht ausgeschrieben.

stellt werden. Andererseits müssen die Modelle der beeinflussten Systemkomponenten in der Lage sein, unterschiedliche Ausfallraten abzubilden. Zur Verbindung zwischen dem aktuellen Systemverhalten und der Ausfallrate der Systemkomponenten werden abstrakte Nutzungsgrade eingeführt. Diese ermöglichen eine Trennung in zwei Bereiche: Die Situationsanalyse und die Reaktion des Systems.

Zur Abbildung lastabhängiger Ausfallraten in den Modellen der Systemkomponenten wird der mögliche Wertebereich der jeweiligen Ausfallraten diskretisiert. Für jeden Teilbereich wird nun eine bedingte Transition in der `Behavior`-Definition des Systemelements notwendig, die neben der notwendigen Bedingung die Rate der zugehörigen Exponentialverteilung enthält. Somit ergibt sich ein Modell mit mehreren parallel verlaufenden Transitionen, siehe auch Bild 2.

Bei der Situationsanalyse wird das Systemverhalten in mehrere Nutzungsgrade eingeteilt. Das System reagiert über `condition`- und `forward`-Statements, indem in jedem Aktor die zugehörige `FailureRate` aktiviert wird. Dies geschieht in der Konfigurationssteuerung und ist somit Teil des Operator-Controller-Moduls (OCM).

4.2.3 Modellierung des Operator-Controller-Moduls und des Mehrstufigen Verlässlichkeitskonzepts

Sämtliche selbstoptimierungsspezifischen Aspekte des zu modellierenden Systems werden im LARES-Modell, analog zum realen System, im OCM zusammengefasst.

Zur Nachbildung der situationsabhängigen Belastung der Aktoren werden alle möglichen Betriebszustände in Nutzungsgrade eingeteilt. Die Betriebszustände wiederum ergeben sich durch die aktuelle Umgebung, die Anforderungen und die Klassifizierung des Mehrstufigen Verlässlichkeitskonzepts und werden durch die Konfigurationssteuerung eingestellt. Sie stellt somit einen zentralen Punkt des OCM dar und wird entsprechend detailliert modelliert. In ihr wird jede mögliche Reglerkonfiguration als Zustand mit den zur Umschaltung notwendigen Transitionen abgebildet. Dadurch ergibt sich beispielhaft für das Feder-Neige-Module der in Bild 3 gezeigte Zustandsraum.

Die tatsächliche Reglerkonfiguration ist für die Evaluation des Modells nicht relevant; es wird lediglich der zugehörige Nutzungsgrad benötigt. Diese Abstrahierung wird über die Formulierung entsprechender `Conditions` in der Moduldefinition der Konfigurationssteuerung `CC` erreicht, die den in Bild 3 markierten Bereichen entsprechen:

```
Condition low_usage = CC.nominal_dlow
Condition medium_usage = CC.nominal_dhigh | CC.nominal_dmed |
   CC.CtrlRec_dmed | CC.CtrlRec_dlow
Condition high_usage = CC.CtrlRec_dhigh | CC.Inactive
```

Die aktuelle Reglerkonfiguration ergibt sich aus den Anforderungen des Nutzers `Req`, den Umgebungsbedingungen `Env` und dem Systemzustand, genannt `MLDC`, selbst.

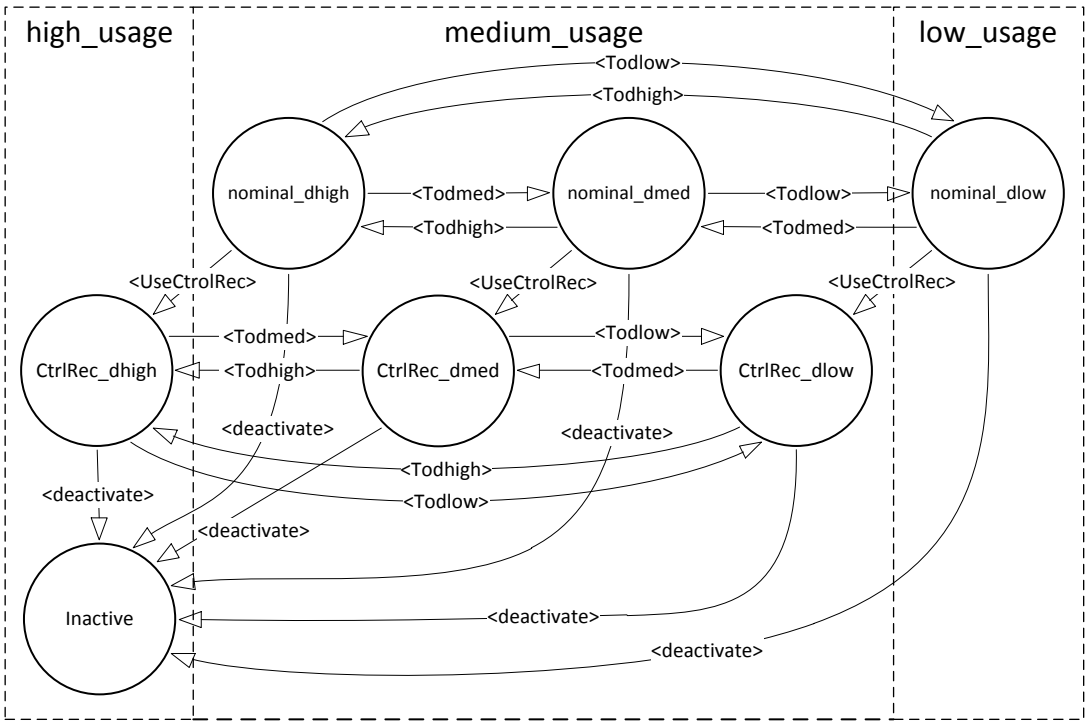

Bild 3: Modell der Konfigurationssteuerung CC. Den Zuständen sind die Nutzungsgrade high_usage, medium_usage und low_usage zugeordnet.

Dieser wird durch das Mehrstufige Verlässlichkeitskonzept bestimmt, das zu diesem Zweck über Kenntnis des Zustands aller zu überwachenden Systemkomponenten verfügt. Die Konfiguration wird über die Nutzung entsprechender `Guards` umgesetzt:

```
Env.Rough_Track & Req.comfort guards CC.<Todhigh>
Env.Rough_Track & Req.balanced | Env.Smooth_Track & Req.comfort
   guards CC.<Todmed>
Env.Rough_Track & Req.energy | Env.Smooth_Track & Req.energy guards
   CC.<Todlow>
MLDC.reconfigurable guards CC.<UseCtrlRec>
( MLDC.stabilizable | MLDC.failed ) guards CC.<deactivate>
```

Die Umgebung und die Anforderungen haben somit maßgeblichen Einfluss auf das Systemverhalten.

4.2.4 Modellierung der Umgebung und der Anforderungen

Sowohl Umgebungs- als auch Anforderungsmodell erhalten bei diesem speziellen Modell keine nennenswerte Rückwirkung vom System und werden daher in sich abgeschlossen modelliert. Es werden dazu qualitativ beschriebene Situationen gewählt. Zu jeder Situation wird ein Zustand definiert, von dem aus in jeden anderen Zustand gewechselt werden kann. Es obliegt nun dem Entwickler, abzuschätzen, wie lange eine jede Situation anhalten wird, und die Transitionen entsprechend zu parametrieren. Der sich ergebende Zustandsraum ist exemplarisch für das Modell der Anforderungen in Bild 4 dargestellt.

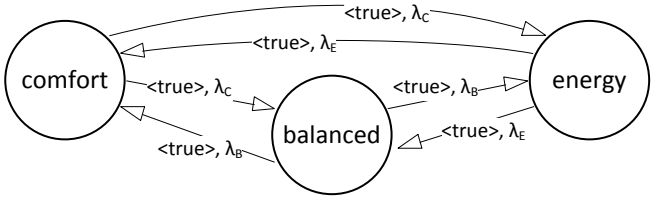

Bild 4: *Modell der Anforderungen mit drei exemplarischen Situationen „comfort“: komfortorientiert, „energy“: energiesparend, „balanced“: Kompromiss zwischen komfortorientiert und energiesparend. Jede Situation entspricht einer Auswahl und Priorisierung der Ziele.*

Durch die Interaktion der Komponenten des OCM wird somit über den Nutzungsgrad in Abhängigkeit der aktuellen Situation aus den Modellen der Umgebung und der Anforderungen sowie des Zustands des Systems die Ausfallrate der Zylindermodule gesetzt.

4.3 Ergebnisse

Das mit der Modellierungssprache LARES erstellte Modell kann nun ausgewertet werden. Als Ausgabemaß wurde dabei die Wahrscheinlichkeit des Erreichens der vier Zustände des Mehrstufigen Verlässlichkeitskonzepts gewählt. Wir gehen dabei davon aus, dass das System in den Stufen 1 und 2 nutzbar, in den Stufen 3 und 4 hingegen nicht mehr nutzbar ist. Es ergibt sich der in Bild 5 gezeigte Wahrscheinlichkeitsverlauf.

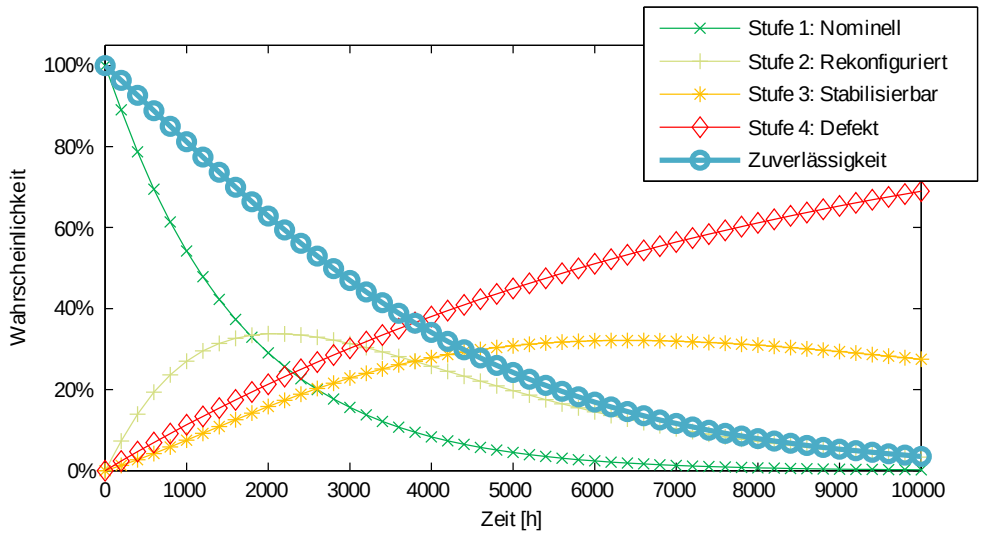

Bild 5: *Verlauf der Zuverlässigkeit und der Wahrscheinlichkeiten der vier Systemzustände*

Es zeigt sich, dass das System zu Beginn des dargestellten Zeitraums einwandfrei nutzbar ist, jedoch bereits nach wenigen tausend Betriebsstunden die Wahrscheinlichkeit des zweiten Zustands dominiert. In diesem Zustand sind ein, maximal zwei Zylindermodule

im Fehlermode *‚stuck‘* ausgefallen, was zwar eine Einschränkung der Funktionsfähigkeit mit sich zieht, jedoch noch nicht als Systemausfall klassifiziert wird. Nimmt die Anzahl derart ausgefallener Zylindermodule zu, oder fallen zu viele Beschleunigungssensoren aus, ist das System zwar noch stabilisier- aber nicht mehr nutzbar. In dieser Phase dominiert die Wahrscheinlichkeit, dass das System im Zustand 3 ist. Sollte mindestens ein Zylindermodul mit dem Fehlermode *‚loose‘* ausfallen, oder fällt ein 3D-Positionssensor aus, ist das System überhaupt nicht mehr nutzbar; dies entspricht dem vierten Zustand. Die Zuverlässigkeit ergibt sich als Summe der Wahrscheinlichkeiten der Zustände 1 und 2. Wie man anhand von Bild 5 sieht, wird durch die Berücksichtigung der Rekonfigurationsfähigkeit und der Verhaltensadaption eine deutlich höhere Zuverlässigkeit erreicht als für Zustand 1, der bereits beim Ausfall einer beliebigen Komponente nicht mehr erreicht wird.

5 Zusammenfassung

Um eine Bewertung der Zuverlässigkeit selbstoptimierender Systeme vornehmen zu können, ist es notwendig, nicht nur das System selbst, sondern auch die Interaktion zwischen den auf das System wirkenden äußeren Bedingungen, der Systemanpassung und den Auswirkungen auf das Ausfallverhalten der einzelnen Systemkomponenten zu modellieren. Zu diesem Zwecke ist eine Modellierungssprache notwendig, die in der Lage ist, diese Aspekte mit einem ausreichend hohen Detaillierungsgrad abzubilden.

Die Modellierungssprache LARES und das zugehörige Framework können dazu genutzt werden, alle notwendigen Aspekte selbstoptimierender mechatronischer Systeme in einem umfassenden Modell abzubilden und somit die sich stellenden Probleme zu lösen. Durch eine Evaluation des Modells ist es möglich, präzise Aussagen über die Zuverlässigkeit des Systems zu machen.

Die aktuelle sprachliche Mächtigkeit von LARES entspricht derjenigen von Markov-Ketten, d.h. die Verteilungen sind exponentiell und als exhaustives Verfahren kann die numerische Analyse an einer Explosion des Zustandsraumes scheitern. Ein Ausweg wären hier simulative Ansätze. Abhängig von den Anforderungen an eine Modellierung bzw. an die Bewertung, z.B. hybrider Systeme oder Prüfung temporaler Eigenschaften, müssen sprachliche Erweiterungen gemacht, bzw. andere Analyseansätze wie z.B. Model-Checking eingesetzt werden.

6 Danksagung

Teile dieses Beitrags wurden im Rahmen des Sonderforschungsbereichs 614 „Selbstoptimierende Systeme des Maschinenbaus“ von der Deutschen Forschungsgemeinschaft (DFG) gefördert. Die Entwicklung von LARES wurde im Rahmen des Projekts SI 710/7-1 durch die DFG gefördert.

Literatur

[BCC+10] BOZZANO, M., CAVADA, R., CIMATTI, A., KATOEN, J.-P., NGUYEN, V. Y., NOLL, T., OLIVE X.: Formal verification and validation of AADL models. In: Proc. of Embedded Real Time Software and Systems Conference (2010).

[BCR+09] BOZZANO, M., CIMATTI, A., ROVERI, M., KATOEN, J.-P., NGUYEN, V.Y., NOLL, T.: Codesign of Dependable Systems: A Component-Based Modeling Language. In: 7th IEEE/ACM International Conference on Formal Methods and Models for Co-Design, 2009.

[Bir07] BIROLINI, A.: Reliability Engineering – Theory and Practice. Springer, 2007.

[BRS+09] BACHMANN, J., RIEDL, M., SCHUSTER, J., SIEGLE, M.: An Efficient Symbolic Elimination Algorithm for the Stochastic Process Algebra Tool CASPA. In: SOFSEM '09: Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science, Berlin, Heidelberg, S. 485–496. Springer LNCS 5404.

[DBB92] DUGAN, J. B., BAVUSO, S. J., BOYD, M. A.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Transactions on Reliability, Band 41, Nummer 3, 1992.

[DIN61025] DIN EN 61025: Fehlzustandsbaumanalyse (IEC 61025:2006), Deutsche Fassung EN 61025:2007, August 2007.

[GPK+11] GRIFFAULT, A., POINT, G., KUNTZ, F., VINCENT, A.: Symbolic computation of minimal cuts for AltaRica models. Research Report RR-1456-11, 2011.

[GRS+12] GOUBERMAN, A., RIEDL, M., SCHUSTER, J., SIEGLE, M.: A Modelling and Analysis Environment for LARES. In: SCHMITT, J. B. (Hrsg.), MMB/DFT, Lecture Notes in Computer Science, Band 7201, S. 244-248. Springer, 2012.

[GRS09a] GAUSEMEIER, J., RAMMIG, F., SCHÄFER, W. (Hrsg.): Verlässlichkeit selbstoptimierender Systeme. HNI-Verlagsschriftenreihe, Band 235, Paderborn, 2009.

[GRS09b] GAUSEMEIER, J., RAMMIG, F., SCHÄFER, W. (Hrsg.): Selbstoptimierende Systeme des Maschinenbaus. HNI-Verlagsschriftenreihe, Band 234, Paderborn, 2009.

[MKS+13] MEYER, T, KEßLER, J. H., SEXTRO, W. TRÄCHTLER, A.: Increasing Intelligent Systems' Reliability by using Reconfiguration. In: Proc. Annual Reliability and Maintainability Symposium (RAMS) 2013.

[MT95] MALHOTRA, M., TRIVEDI, K. S.: Dependability modeling using Petri-nets. In: IEEE Transaction on Reliability, Band 44, Nummer 3, S. 428-440, 1995.

[PR99] POINT, G., RAUZY, A.: AltaRica - Constraint automata as a description language. In: European Journal on Automation, Ausgabe 33(8-9), S. 1033-1052, Hermes, 1999.

[RS12] RIEDL, M., SIEGLE, M.: A LAnguage for REcongurable dependable Systems: Semantics & Dependability Model Transformation. In: Proc. Sixth International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS 2012), S. 78-89, CNAM. Paris, France, 2012.

[SS10] SONDERMANN-WÖLKE, C., SEXTRO, W.: Integration of Condition Monitoring in Self-optimizing Function Modules Applied to the Active Railway Guidance Module. In: International Journal on Advances in Intelligent Systems, Band 3, Nummer 1-3, S. 65-74, 2010.

[VDI4008] VDI 4008: Markoff-Zustandsänderungsmodelle mit endlich vielen Zuständen, 1999.

[WGR+09] WALTER, M., GOUBERMAN, A., RIEDL, M., SCHUSTER, J., SIEGLE, M.: LARES - A Novel Approach for Describing System Reconfigurability in Dependability Models of Fault-Tolerant Systems. In: Proc. of European Safety and Reliability Conference (ESREL'09), S. 153-160, Taylor & Francis 2009.

Autoren

Tobias Meyer hat Maschinenbau an der Universität Paderborn studiert. Seit 2011 ist er wissenschaftlicher Mitarbeiter am Lehrstuhl für Mechatronik und Dynamik der Universität Paderborn. Im Rahmen des Sonderforschungsbereich 614 ‚Selbstoptimierende Systeme des Maschinenbaus' befasst er sich vorwiegend mit der Verlässlichkeit selbstoptimierender Systeme.

Christoph Sondermann-Wölke studierte Ingenieurinformatik an der Universität Paderborn. Seit 2007 arbeitet er am Lehrstuhl für Mechatronik und Dynamik der Universität Paderborn als wissenschaftlicher Mitarbeiter. Innerhalb des Sonderforschungsbereichs 614 ‚Selbstoptimierende Systeme des Maschinenbaus' beschäftigt er sich mit Methoden zur Steigerung der Verlässlichkeit mechatronischer Systeme.

Walter Sextro hat Maschinenbau an der Leibniz Universität Hannover und am Imperial College in London studiert. Nach seiner Industrietätigkeit in Deutschland und den USA promovierte er 1997 an der Universität Hannover. Seine Habilitation hat er im Bereich dynamischer Kontaktprobleme mit Reibung verfasst. In den Jahren 2004-2009 hatte er eine Professur an der Technischen Universität Graz inne. Seit 2009 leitet er den Lehrstuhl für Mechatronik und Dynamik der Universität Paderborn.

Martin Riedl studierte Informatik mit Schwerpunkt Syntaxanalyse und Compilerbau an der Friedrich-Alexander Universität Erlangen-Nürnberg. Seit 2007 arbeitet er am Institut für Technische Informatik in der Gruppe "Entwurf von Rechen- und Kommunikationssystemen" als wissenschaftlicher Mitarbeiter. Neben der Definition von LARES beschäftigt er sich vorwiegend mit der Weiterentwicklung des zugehörigen Transformations- und Analyseframeworks.

Alexander Gouberman studierte Diplom-Mathematik mit Schwerpunkt Differentialgeometrie an der Universität Augsburg. Seit 2008 arbeitet er am Institut für Technische Informatik in der Gruppe "Entwurf von Rechen- und Kommunikationssystemen" als wissenschaftlicher Mitarbeiter. Seine Forschungsinteressen umfassen u.a. die Bewertung und Optimierung der Zuverlässigkeit und Performanz von Systemen.

Markus Siegle studierte Informatik mit Nebenfach Elektrotechnik an der Universität Stuttgart. Anschließend war er als Stipendiat der Fulbright-Stiftung an der North Carolina State University, wo er ein Masters Degree erwarb. Promotion und Habilitation erfolgten am Lehrstuhl für Rechnernetze und Kommunikationssysteme der Friedrich-Alexander-Universität Erlangen-Nürnberg. Seit 2003 ist Markus Siegle Inhaber der Professur für Entwurf von Rechen- und Kommunikationssystemen (Fakultät für Informatik) der Universität der Bundeswehr München. Zu seinen Arbeitsschwerpunkten gehört die Analyse der Leistungsfähigkeit und Zuverlässigkeit von IT-Systemen mit Hilfe stochastischer Modelle.