

# Eine erweiterte Systemmodellierung zur Entwicklung von softwareintensiven Anwendungen in der Automobilindustrie\*

Jörg Holtmann, Jan Meyer, Wilhelm Schäfer und Ulrich Nickel

chrome|janny|wilhelm@upb.de

Ulrich.Nickel@hella.com

**Abstract:** Heutige Transportsysteme, wie z.B. Automobile sind gekennzeichnet durch eine Vielzahl von Funktionalität, die sehr häufig durch Software realisiert wird. Hiermit ist eine Zunahme der Komplexität festzustellen. Zur Beherrschung dieser Komplexität und damit einhergehend die Aufteilung des Systems in verschiedene Komponenten, ist eine Gesamtmodellierung des Systems inklusive des Verhaltens unerlässlich. Ein besonderer Augenmerk liegt auf Grund der Echtzeitsysteme in dieser Domäne dabei auf der Modellierung von Zeiten auf Systemebene. Die derzeitigen Modellierungskonzepte, wie beispielsweise die Systems Modeling Language (SysML), sind dafür aber noch nicht ausreichend. In dem hier vorgestellten Ansatz wird eine erweiterte Systemmodellierung vorgestellt, die zusätzlich eine formale Spezifizierung von Zeiten erlaubt. Durch diese Modellierung sind weitere Analysemethoden, wie z.B. Simulationen oder Verifikationen möglich, die zum einen die sicherheitsrelevante Funktionalität sicherstellen und zum anderen die Qualität der Software steigern.

## 1 Einleitung

Eingebettete Systeme sind nicht wegzudenkende Bestandteile des täglichen Lebens. Sie lassen sich in einfachen Haushaltsgeräten bis hin zu komplexen Transportsystemen etc. finden. Bei der Entwicklung aller eingebetteten Systeme treten vergleichbare Probleme auf. So werden beispielsweise immer mehr Sensoren und bestehende Systeme miteinander vernetzt, um eine möglichst genaue Steuerung der Systeme zu ermöglichen. Die hiermit einhergehende Komplexität, insbesondere bei der zeitlichen Spezifikation des Verhaltens, erschwert die Entwicklung. Zur Darstellung dieser Problemstellung und einer entsprechenden Lösung wird hier stellvertretend auf die Entwicklung eines eingebetteten Systems aus dem Automobilbereich eingegangen.

Die Vielzahl von Innovationen im Automobilbereich, welche zu großen Anteilen durch Software realisiert werden, lässt die Komplexität des Gesamtsystems „Kraftfahrzeug“ bekanntermaßen ansteigen [Gri03]. Grund hierfür ist die hohe Anzahl von Steuergeräten

---

\*Diese Arbeit entstand im Rahmen des SPES2020 Projektes gefördert durch das Bundesministerium für Bildung und Forschung (BMBF), "SPES2020,01IS08045H"

(Embedded Control Units - ECUs), welche über heterogene Netzwerke miteinander kommunizieren und letztendlich ein komplexes Netzwerk von Funktionen realisieren. Betrachtet man die einzelnen ECUs eines Fahrzeugs näher, so ist auch hier ein Zuwachs an Komplexität zu erkennen. Leistungsfähigere Hardware ermöglicht die Integration von Funktionen auf einer ECU, welche vormals auf mehrere ECUs verteilt waren. Darüber hinaus machen einige Funktionen, z.B. Fahrerassistenzsysteme, welche auf einer Echtzeit-Bilddatenverarbeitung basieren, zudem den Einsatz mehrerer Rechenkerne (Mikrocontroller, DSPs oder FPGAs) unerlässlich.

Hieraus ergeben sich neue Problemstellungen, die es zu lösen gilt. Zunächst müssen die einzelnen Funktionen möglichst optimal auf die jeweiligen Rechenkerne aufgeteilt werden. Somit ergibt sich ein, für die Performance wichtiges, Optimierungsproblem. Die Zuordnung von unterschiedlichsten Funktionen auf einen Rechenkern, die sich ggf. noch die Rechenleistung gegenseitig wegnehmen, während gleichzeitig ein anderer Rechenkern nicht voll ausgelastet ist, spricht dafür, dass das System insgesamt nicht optimal implementiert ist. Eine weitere Vorgabe bei der steigenden Funktionalität ist die Sicherstellung des korrekten Aufrufs der Funktionen, der Bereitstellung der Daten und der Kommunikation zwischen ihnen [Bro06]. Diese Problemfelder müssen frühzeitig bei der Entwicklung erkannt und überprüft werden. Spätere Anpassungen können größere Umstrukturierungen und somit auch höhere Kosten verursachen. Von daher müssen bereits in frühen Entwicklungsphasen Methoden zur formalen Spezifikation des Systemverhaltens inklusive der zeitlichen Anforderungen bereitgestellt werden. Nur diese bieten dann auch Möglichkeiten für Analysemethoden bieten, beispielsweise um die Verteilung der Funktionen zu unterstützen. Derzeit werden diese Probleme mit dem Erfahrungswissen der Entwickler und aufwendigen Integrationsphasen behoben. Dies ist aber ein kostenintensiver Ansatz.

Eine Besonderheit in der Automobilbranche ist, dass große Teile der Entwicklung an Zulieferer zwecks Realisierung ausgelagert werden. Deshalb werden vom Automobilhersteller (OEM) Anforderungen erstellt und mit dem jeweils beauftragten Zulieferer ausgetauscht [WW03]. Die Anforderungen sind zum Teil noch unvollständig. Dies ist vor allem der Fall, wenn das Wissen und die Experten beim Zulieferer zu finden sind. Alle Anforderungen werden in der Anforderungsanalyse genauer analysiert und ggf. durch das explizite Erfahrungswissen des Zulieferers ergänzt. Aus den Anforderungen wird schließlich ein initiales Systemmodell (Architektur und Verhalten) erstellt. Diese ist die Basis für die weitere Entwicklung und für Designentscheidungen, wie z.B. die Aufteilung der Funktionalität.

Bei einem eingebetteten System in der Automobilindustrie spielt die Echtzeitfähigkeit und somit die Einhaltung der zeitlichen Anforderungen eine wesentliche Rolle. Dies ist besonders bei sicherheitskritischen Funktionen (Airbag, Steuerung, Bremssystem, etc.) notwendig. Die zeitlichen Anforderungen lassen sich auf allen Ebenen und in unterschiedlichsten Detailstufen wiederfinden. Beispiele für die Systemebene werden anhand eines Komfortsteuergerätes in Kapitel 4 vorgestellt. Die für das Gesamtsystem relevanten zeitlichen Anforderungen müssen in die Systemmodellierung integriert werden. Dies bedeutet, dass bei globalen zeitlichen Anforderungen diese auf einzelne Systembestandteile aufgeteilt werden müssen. Dies kann mitunter in Form von Zeitbudgets erfolgen. Damit die Aufteilung unterstützt durch Analysen bzw. automatisch erfolgen kann, muss natürlich das Gesamt-

systemverhalten und auch die zeitlichen Anforderungen formal spezifiziert sein.

Mit den heutigen Modellierungssprachen ist es nur bedingt möglich, eine Systemarchitektur mit gleichzeitiger formaler Spezifikation des Systemverhaltens zu modellieren. Dies wird in Kapitel zwei (Stand der Technik) dargelegt. Es werden die verschiedenen Systemmodellierungssprachen, die derzeit in der Industrie eingesetzt werden, mit ihren Vor- und Nachteilen gegenübergestellt. Aus den dort dargelegten Argumenten und den Problemstellungen aus Kapitel eins ergibt sich, dass die bisherigen Ansätze nicht ausreichen und daher erweitert werden müssen. Von daher wird im dritten Kapitel dieses Papiers ein Ansatz einer erweiterten Systemmodellierung vorgestellt, der eine formale Spezifizierung von Verhalten ermöglicht. Hierfür verwenden wir die Systems Modeling Language (SysML) durch eine formale Zeitmodellierung ergänzt. Der Ansatz ermöglicht darüberhinaus auch weitergehende Analysetechniken, beispielsweise Simulationen oder Model Checking Techniken. Die Ergebnisse der Analyseverfahren können z.B. dazu genutzt werden das Verteilungsproblem der einzelnen Funktionen zu lösen. Im vierten Kapitel wird anhand eines Beispiels die Verwendung des vorgestellten Ansatzes aufgezeigt. Zum Schluss wird eine Zusammenfassung und ein Ausblick gegeben, inwieweit der Ansatz in Zukunft noch ausgebaut werden kann, um auch zukünftige Problemstellungen abzudecken.

## 2 Stand der Technik

Es existieren verschiedene Techniken, um ein softwareintensives System zu modellieren und zu analysieren. In diesem Kapitel werden die gängigsten Modellierungssprachen auf ihre Nutzung bezüglich eingebetteter Systeme hin untersucht. In [GH06] werden verschiedene Modellierungssprachen auf ihre Eignung bezüglich der Modellierung von softwareintensiven Systemen untersucht. Auch die in der Industrie weit verbreiteten Sprachen MATLAB<sup>1</sup> und SysML wurden näher untersucht. Als Ergebnis dieser Studie bleibt festzuhalten, dass die Modellierungssprachen für zukünftige softwareintensive Systeme sowohl die Struktur, aber auch das Verhalten formal spezifizieren müssen. Ebenso ist die Modellierung von Zustandsautomaten mit hybridem Verhalten und Uhren notwendig [GH06]. Damit einhergehend ist eine Anbindung an Analysewerkzeuge, exemplarisch sei hier auf das Model Checking hingewiesen, erforderlich. Unter diesen Gesichtspunkten hat sich gezeigt, dass der Ansatz „Mechatronic UML“ mit den darin enthaltenen Real-Time Statecharts am Besten abschneidet. Deshalb werden diese im Ansatz auch wiederverwendet.

### MATLAB/Simulink

Im Bereich der Regelungstechnik wird in der Industrie sehr häufig das Werkzeug MATLAB/Simulink verwendet. Mit der zu Grunde liegenden Modellierungssprache ist es möglich, kontinuierliche Systeme zu modellieren. Zugleich ist häufig der Ansatz zu finden, die Sprache auch für die Spezifizierung von diskreten Anteilen und sogar ganzen Systemen zu nutzen. Es ist somit möglich hybride Systeme zu entwickeln. Hierbei können auch zeitliche Anforderungen berücksichtigt werden. Diese Modellierungsvariante bietet weiterhin den Vorteil, dass eine Simulation des spezifizierten Verhaltens möglich ist. Daher ist es

---

<sup>1</sup>[www.mathworks.com](http://www.mathworks.com)

grundsätzlich möglich, ein zeitliches Verhalten einer Komponente bzw. eines Systems zu spezifizieren. Es werden aber keine Uhren mit diesem Ansatz unterstützt [GH06]. Jedoch ist das Werkzeug und somit auch die Sprache nicht dafür konzipiert, die Struktur (Architektur) zu erstellen. Dies ist zwar mit verschiedenen Hilfsmethoden durchaus machbar, jedoch entstehen dann sehr unübersichtliche und kaum mehr wartbare Modelle. Von daher ist von der Modellierung der Struktur eines Systems mit MATLAB/Simulink abzuraten.

**Systems Modeling Language (SysML)** Die Systems Modeling Language (SysML) ist eine Modellierungssprache zur Spezifizierung von Systemen und wurde von der Object Management Group (OMG)<sup>2</sup> entwickelt [Sys08]. Die SysML beschreibt die Systeme dabei auf einer abstrakten Art, so dass noch nicht entschieden ist, in welcher Ingenieursdisziplin sie später realisiert werden. So können sowohl Hardware, Software, aber auch mechanische Bestandteile modelliert werden. Sie ist auf Basis der UML 2.0 entwickelt worden. Dabei ist ein Teil der UML, vor allem die Verhaltensmodellierung, übernommen worden. Zur Anpassung an die Systemmodellierung sind zusätzliche Elemente und Diagramme hinzugekommen, wie beispielsweise das Anforderungs- das Blockdefinitions- und das Zuicherungsdiagramm. Im Gegensatz zur UML 2.0 gehören jedoch Diagramme, die nur zur Beschreibung von Software genutzt wurden, wie das Objektdiagramm, nicht zur SysML [Wei08]. Insgesamt ist zur SysML zu sagen, dass ihre Vorteile besonders bei der Modellierung der Architektur zu sehen sind. Es ist zwar auch möglich das Verhalten zu modellieren, aber die SysML bietet nur eingeschränkte Möglichkeiten zur Zeitmodellierung. Es beschränkt sich zum größten Teil auf das When- und das After Konstrukt. Dies ist für die Spezifizierung von Echtzeitsystemen jedoch nicht ausreichend.

**Zusätzliche Profile für die UML/SysML** Die SysML besitzt, wie auch die UML, einen Erweiterungsmechanismus in Form von Profilen. Hierdurch können die Sprachen erweitert und an bestimmte Domänen angepasst werden. Zwei von der OMG spezifizierte Profile befassen sich mit der Performance Modellierung. Diese können auch für eingebettete Systeme genutzt werden. Die beiden Profile sind das Profil für Schedulability, Performance and Time (SPT) [Spt05] und das Nachfolgerprofil Modeling and Analysis of Real-time and Embedded systems (MARTE) [OMG07]. Beide Profile bieten die Möglichkeit der Modellierung von Zeiten, Nebenläufigkeit und Performance. In den Profilen wird aber nirgends ausgesagt, welcher Stereotyp eindeutig für welche Information verwendet werden muss. Somit sind unterschiedlichste Modellierungsformen möglich. Ferner fehlen Aussagen darüber, wie die neuen Informationen in den verschiedenen Verhaltensdiagrammen integriert werden können, beispielsweise in den Zustandsdiagrammen [HH07]. Außerdem bieten sie keine Möglichkeit, mangels einer präzisen semantischen Definition, die Modelle automatisch auf Korrektheit und Konsistenz zu überprüfen. Somit ist eine eindeutige und formale Modellierung nur bei rigiden Modellierungsvorschriften gegeben und die Verwendung der Profile zu einer anschließenden Systemanalyse nur eingeschränkt möglich.

Es existiert auch noch ein an die Automobilindustrie angepasstes Profil der UML. Dies ist die sogenannte EAST-ADL2 [Ead08]. Dies Profil stellt Stereotypen und Eigenschaftswerte zur Verfügung, die typische Elemente aus der Automobilindustrie, wie beispielsweise die verschiedenen Bussysteme, repräsentieren. Zur Beschreibung von Verhalten können verschiedene Modellierungssprachen genutzt werden, wie Anwendungsfälle und Zustandsau-

---

<sup>2</sup>[www.omg.org](http://www.omg.org)

tomaten [HKM07]. Somit können auch dort nur eingeschränkt Zeiten modelliert werden.

**Fazit** Ausgehend von diesen Untersuchungen bleibt festzuhalten, dass bei allen bestehenden Ansätzen Defizite vorhanden sind. Während einige Ansätze besser die Architektur darstellen können, sind andere besser geeignet das Verhalten zu modellieren. Eines ist aber bei allen Ansätzen gemeinsam. Bei der Spezifikation von Zeiten und der Anbindung an Verifikations- und Simulationsumgebungen gibt es noch Handlungsbedarf, denn es fehlt ein geeigneter Formalismus zur Spezifikation von Zeiten.

### 3 SysML ergänzt um Real-Time Statecharts

Wie aus der Motivation zu erkennen ist, bedarf es zur Modellierung von softwareintensiven Systemen in der Automobilindustrie sowohl der Systemarchitektur, als auch des Systemverhaltens. Diese können, wie in Kapitel 2 beschrieben, zurzeit nur unzureichend mit den bisherigen Konzepten modelliert werden. Deshalb wird im Folgenden ein Ansatz für eine Systemmodellierung vorgestellt, der sowohl die Systemarchitektur als auch das Verhalten formal modellieren kann. Wie bereits beschrieben, bietet sich zur Modellierung der Systemarchitektur die SysML an. Vor allem auch, weil sie ein internationaler Standard im Bereich der Systemmodellierung ist. Die angesprochenen Nachteile (Verhaltens- und Zeitmodellierung) lassen sich durch eine Erweiterung der SysML beheben. Hierzu werden die sogenannten Real-Time Statecharts (RTSC) verwendet [GB03] [GTB<sup>+</sup>03].

Die Real-Time Statecharts wurden im Sonderforschungsbereich 614 „Selbstoptimierende Systeme des Maschinenbaus“ [SFB09] entwickelt. Der dort entwickelte Ansatz ermöglicht neben der Modellierung der Struktur auch die Modellierung des Verhaltens von selbstoptimierenden mechatronischen Systemen. Für die Spezifikation des Echtzeitverhaltens werden Real-Time Statecharts genutzt. Diese definieren zeitbehaftete UML Zustandsdiagramme auf der Grundlage von Timed Automata. Sie sind somit formal definiert. Im Gegensatz zu den UML Zustandsdiagrammen, bringen die Real-Time Statecharts die Möglichkeit zur Modellierung von Zeiten, wie zum Beispiel Uhren, Worst Case Execution Times (WCET) etc. mit. Die RTSCs vereinen die Konstrukte der Statecharts mit der Semantik der Timed Automata an sich [GB03].

Im folgenden Absatz werden die Möglichkeiten der Real-Time Statecharts dargestellt. In der Abbildung 1 ist ein Beispiel eines Real-Time Statecharts zu sehen, welches aus zwei Zuständen besteht. Das Beispiel dient dazu die verschiedenen Modellierungsmöglichkeiten zu erläutern, die mit einem Real-Time Statechart möglich sind, wobei der letzte Zustand nicht mehr verlassen werden kann. Der Zustand S1 besitzt die Zeitinvariante  $t_0 \leq 5$  ms. Bei Eintritt in diesen Zustand wird die Uhr  $t_0$  auf 0 zurückgesetzt und die Operation `entryS1()` mit der WCET  $w = 1$  ms ausgeführt. Während des Aufenthaltes in S1 wird die Methode `doS1()` mit einer WCET von  $w = 1$  ms und einer Periode, die zwischen 2ms und 3ms liegt, ausgeführt. Die dargestellte Transition schaltet, wenn das Ereignis `e` anliegt, der Guard  $x \leq 2$  und der Zeitguard  $1ms \leq t_0$  wahr sind, wobei `x` eine beliebige Integer Variable ist. Beim Schaltvorgang wird als Seiteneffekt `action()` ausgeführt, der eine WCET

von  $w = 2ms$  besitzt. Er muss spätestens 10ms nach Transitionsaktivierung, aber auch zum Zeitpunkt  $t_1 = 6ms$  beendet sein. Der Seiteneffekt darf frühestens bei  $t_1 = 3ms$  beendet werden. Zum Aktivierungszeitpunkt der Transition wird die Uhr  $t_2$  zurückgesetzt. Die Invariante des Zustands S2 lautet  $t_0 \leq 20ms$  und  $t_1 \leq 13ms$ . Bei Verlassen des Zustands werden die Uhren  $t_0$  und  $t_1$  wieder zurückgesetzt.

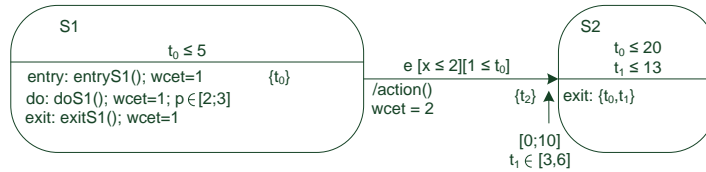


Abbildung 1: Beispiel eines Real-Time Statecharts [Bur02]

Nachdem anhand des Beispiels in Abbildung 1 die Möglichkeiten aufgezeigt wurden, die mit einem Real-Time Statechart möglich sind, wird im Folgenden ein Konzept beschrieben, wie die Statecharts der SysML erweitert werden können, um die zeitlichen Konzepte der Real-Time Statecharts umzusetzen. Diese ermöglichen später eine Analyse der vorgenommenen Informationen. Die Erweiterungen der Real-Time Statecharts, die sich mit der Modellierung von hybriden Systemen beschäftigen, werden dabei zunächst außer Acht gelassen, da sie für die frühen Modellierungsphasen nicht benötigt werden [GBS04].

In der SysML wird zur Spezifikation des Verhaltens u.a. das Zustandsdiagramm der UML 2 übernommen. Darum ist dieses Metamodell zu untersuchen, um Erweiterungspunkte zu finden, an denen die Konzepte der Real-Time Statecharts anknüpfen können. Vergleicht man die beiden Metamodelle von UML- und Real-Time Statecharts, so bleibt festzuhalten, dass die RTSCs alle - mit Ausnahme des After- und des When-Konstruktes - Eigenschaften der UML Statecharts aufweisen. Die Aussagen dieser beiden Konstrukte ist aber implizit in den Real-Time Statecharts gegeben. Jedoch sind sie in zusätzlichen Elementen verborgen, die eine darüber hinausgehende Spezifikation von Zeit erlauben. Von daher lassen sie sich nach modellieren.

Ein RTSC Zustand erweitert einen Zustand der UML um folgende Konstrukte: Zeitinvarianten, Uhren-Resets, WCETs, und ein Periodenintervall. Aber nicht nur der Zustand muss erweitert werden. Auch die UML Transitionen müssen im Metamodell ergänzt werden und zwar um die folgenden Konstrukte: Zeitguards, Uhren-Resets, Prioritäten, Deadlines, WCETs und Synchronisationskanäle mit Synchronisationsarten.

Ausgehend von diesen Ergänzungen des SysML bzw. des UML Zustandsdiagramms ergibt sich das folgende erarbeitete Metamodell (vgl. Abbildung 2) eines erweiterten Zustandsdiagramms, in dem auch die Möglichkeit zur Modellierung von Zeiten und verteilter Kommunikation möglich ist.

In der Abbildung 2 ist das erweiterte Metamodell der SysML bzw. der UML zu sehen. In diesem sind die eben herausgearbeiteten notwendigen Erweiterungen zu sehen, wie exemplarisch die Invarianten [MH09]. Die neu hinzugekommenen Elemente sind dabei gelb (grau schraffiert) und die neuen Assoziationen sind dicker und in grün dargestellt.

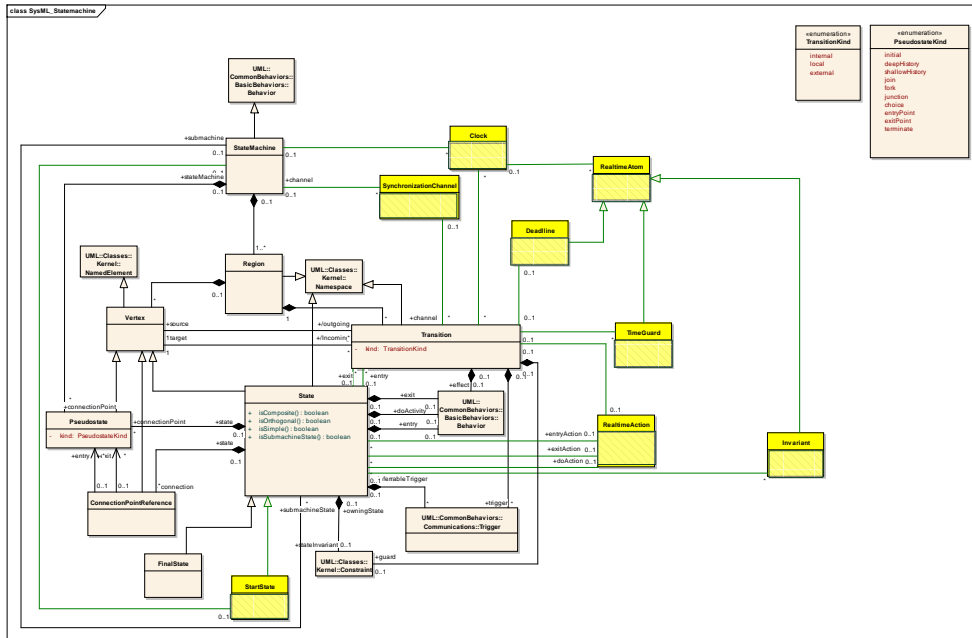


Abbildung 2: Erweitertes MetaModell des UML Zustandsdiagramms

Die Erweiterungen beziehen sich hauptsächlich auf die Zeitmodellierung, wie z.B. durch Uhren etc. Mit diesem erweiterten Metamodell ist die Kombination von SysML und Real-Time Statecharts möglich.

#### 4 Verwendung des Ansatzes

In dieser Ausarbeitung wird als Beispiel für die Benutzung des eben vorgestellten Ansatzes von SysML mit einer Erweiterung von Real-Time Statecharts ein Komfortsteuergerät verwendet. Die Hauptaufgabe eines solchen Steuergerätes ist die Überwachung der Zentralverriegelung inklusive Empfang und Auswertung von Signalen der Funkfernbedienung. Die Funktionalität ist in den letzten Jahren jedoch deutlich gesteigen. So verwaltet ein Komfortsteuergerät auch die Diebstahlwarnanlage und wird für das Öffnen und Schließen der Fenster bzw. des Schiebedaches eingesetzt. Ebenso ist es für die Ansteuerung der Innenbeleuchtung und der Ausstiegswarnleuchten verantwortlich. Dabei ist auch grundsätzlich eine Kombination der verschiedenen Funktionalität möglich. Zur Darstellung, des eben beschriebenen Konzeptes, werden die fünf nachfolgenden Anforderungen verwendet:

1. Das Komfortsteuergerät regelt das Innenlicht beim Öffnen bzw. Schließen der Türen.

2. Nach Öffnen einer Tür muss das Innenraumlicht innerhalb von 50 ms hoch gedimmt werden und für 5 Sekunden leuchten.
3. Nach dem Schließen aller Türen wird das Innenraumlicht nach 2 Sekunden ausgeschaltet.
4. Das Innenraumlicht bleibt höchstens 10 Sekunden an.
5. Falls beim Öffnen einer Tür das Innenraumlicht bereits brennt, so wird die Dauer um 5 Sekunden verlängert.

Diese Anforderungen müssen in die erweiterte Systemmodellierung übernommen werden. Die Anforderung eins wird in der SysML durch einen Block namens „Komfortsteuergerät“ realisiert. Die Abbildung 3 zeigt das aus den Anforderungen 2 - 5 modellierte Real-Time Statechart, welches das Verhalten des Blocks spezifiziert. Es besteht aus zwei Zuständen und vier Transitionen. Die zeitlichen Informationen aus den Anforderungen lassen sich in den Uhren bzw. der Invariante wiederfinden. So findet man beispielsweise die Anforderung 4 in der Zeitinvariante  $Licht\_gesamt \leq 10000$  wieder. Während die Anforderung 5 durch die Transition „Tür geöffnet“ beim Zustand „Innenraumlicht\_an“ repräsentiert wird. Die Anforderung 2 und 3 werden durch die Zeitguards bzw. die Perioden  $[0;50]$  bzw.  $[2000;2005]$  umgesetzt. Der Dimm Vorgang wird dabei innerhalb der Funktion „aktiviereLicht“ behandelt. Bei der letzten Periode ist ein Zeitpuffer von 5ms hinzugefügt worden.

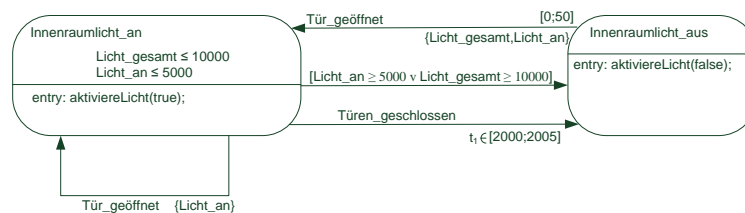


Abbildung 3: Real-Time Statechart der Anforderungen an das Komfortsteuergerät

Das formal spezifizierte zeitliche Verhalten kann nun weiter verwendet werden. So kann mittels formaler Verifikationstechniken (Model Checking) überprüft werden, ob das Komfortsteuergerät die Sicherheitseigenschaften erfüllt, z.B. das keine Verklemmung (Deadlock) vorliegt [Gie05]. Dabei kann nicht nur das Verhalten der Komponente an sich, sondern auch die Kommunikation zwischen ihnen mittels der Real-Time Statecharts beschrieben werden [BGS05]. Hierdurch können die zeitlichen Anforderungen überall formal spezifiziert werden. Ebenso können die Daten auch genutzt werden, um sie innerhalb einer Simulationsumgebung weiter zu analysieren. Hierdurch lassen sich unter anderem Zeitüberschreitungen feststellen. Dafür können beispielsweise Werkzeuge wie ChronSim<sup>3</sup> oder SymTA/S<sup>4</sup> [HHJ<sup>+</sup>05] eingesetzt werden, um die verschiedenen Zeitanforderungen und ihre Aufteilung zu analysieren und zu überprüfen. Die Analyseergebnisse dienen als

<sup>3</sup>www.inchron.de

<sup>4</sup>www.symtavision.com

Grundlage für spätere Designentscheidungen. So kann bei dem eben vorgestellten Komfortsteuergerät beispielsweise die CPU-Last analysiert werden. Wenn sie zu hoch ist, muss ggf. eine leistungsstärkere Hardware genutzt werden. Ebenso kann durch die Analyseergebnisse entschieden werden, ob die Funktion zur Steuerung des Innenlichts auf mehrere Rechenkerne verteilt werden kann. Die Ergebnisse ergänzen somit das Erfahrungswissen der Entwickler, was bisher ausschließlich die Grundlage für Designentscheidungen war. Die Erweiterung ist exemplarisch umgesetzt und wird derzeit im Rahmen des SPES2020 Projektes weiter evaluiert [SPE09].

## 5 Resümee und Ausblick

Bei eingebetteten Systemen und vor allem bei Systemen im Automobilbereich hat die Einhaltung von zeitlichen Anforderungen eine wesentliche Bedeutung. Auf Grund der steigenden Funktionalität und der damit einhergehenden steigenden Komplexität muss bereits in den frühen Entwicklungsphasen dafür Sorge getragen werden, dass diese Anforderung beachtet und erfüllt werden. Dies wird zukünftig immer schwieriger, wenn nicht geeignete Modellierungsmöglichkeiten zur Verfügung stehen, um diese Komplexität beherrschbar zu gestalten.

Mit dem hier vorgestellten Ansatz einer Systemmodellierung mit SysML - erweitert durch Real-Time Statecharts - ist es nun möglich, das System- und Kommunikationsverhalten inklusive zeitlicher Bedingungen formal zu spezifizieren. Hierdurch bekommt der Entwickler sowohl einen besseren Überblick als auch die Möglichkeit sein Modell zu analysieren und zu überprüfen. Dies kann sowohl durch Simulationen als auch durch Model Checking erfolgen. Das Model Checking kann auch bei größeren Systemen durchgeführt werden, da die Real-Time Statecharts eine Dekomposition ermöglichen. Hierdurch kann bereits in frühen Entwicklungsphasen das Modell verifiziert werden. Fehler und Intoleranzen werden frühzeitig erkannt und treten nicht erst im Integrationstest auf. Dementsprechend können sie früh und ohne großen Ressourcenaufwand behoben werden. Dies ist eine deutliche Verbesserung der jetzigen Entwicklung und in der Zukunft auch notwendig.

Die Anbindung des hier vorgestellten Ansatzes an einen Model Checker ist bereits möglich [GBS04]. Eine vollständig automatisierte Anbindung an eine Simulationsumgebung ist jedoch noch nicht gegeben. Es existieren bereits erste Ansätze und Prototypen, die noch ergänzt werden müssen. Hierdran wird aber im SPES2020 Projekt [SPE09] weiter geforscht. Denn ein Großteil der Informationen, die für eine Simulation notwendig ist, ist bereits im erweiterten Systemmodell vorhanden. Somit ist eine passende Abbildung zu definieren, um eine semi-automatische Wiederverwendung zu ermöglichen. Dieser Automatisierungsschritt bedeutet auf der einen Seite eine schnellere Anbindung an die Simulationsumgebung ohne das redundante Datenbestände notwendig sind. Auf der anderen Seite wird so auch eine Fehlerquelle, nämlich das manuelle Übertragen von Informationen, ausgeschaltet.

## Literatur

- [BGS05] Sven Burmester, Holger Giese und Wilhelm Schäfer. Model-Driven Architecture for Hard Real-Time Systems: From Platform Independent Models to Code. *Model Driven Architecture Foundations and Applications ECMDA-FA*, 2005.
- [Bro06] Manfred Broy. Challenges in Automotive Software Engineering. *International Conference on Software Engineering (ICSE)*, 2006.
- [Ead08] EAST-ADL2 Specification, 2008.
- [GB03] Holger Giese und Sven Burmester. Real-Time Statechart Semantics. Bericht, Universität Paderborn, 2003.
- [GBS04] Holger Giese, Sven Burmester und Wilhelm Schäfer. Modular Design and Verification of Component-Based Mechatronic Systems with Online-Reconfiguration. *Foundations of Software Engineering (FSE)*, 2004.
- [GH06] Holger Giese und Stefan Henkler. A survey of approaches for the visual model-driven development of next generation software-intensive systems. *Journal of Visual Languages & Computing*, 2006.
- [Gie05] Holger Giese. Modeling and Verification of Cooperative Self-adaptive Mechatronic Systems. *Monterey Workshop*, 2005.
- [Gri03] Klaus Grimm. Software Technology in an Automotive Company - Major Challenges. *25th International Conference on Software Engineering*, 2003.
- [GTB<sup>+</sup>03] Holger Giese, Matthias Tichy, Sven Burmester, Wilhelm Schäfer und Stephan Flake. Towards the Compositional Verification of Real-Time UML Designs. *Proceedings of the 9th European software engineering conference*, 2003.
- [HH07] Matthias Hagner und Michaela Huhn. Modellierung und Analyse von Zeitanforderungen basierend auf der UML. *GI-Workshop: Automotive Software Engineering*, 2007.
- [HHJ<sup>+</sup>05] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter und Rolf Ernst. System Level Performance Analysis - the SymTA/S Approach. *IEEE Proceedings of Computers and Digital Techniques*, 152, 2005.
- [HKM07] Martin Hobelsberger, Stefan Kuntz und Jürgen Mottok. Architekturmodellierung: Vergleich von EAST-ADL und SAE AADL. *Automotive*, Hanser Verlag, 2007.
- [MH09] Jan Meyer und Jörg Holtmann. SPES Deliverable AU-D.3.3.A, 2009.
- [OMG07] OMG. UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE), 2007.
- [SFB09] Sonderforschungsbereich 614 "Selbstoptimierende Systeme des Maschinenbaus", [www.sfb614.de](http://www.sfb614.de), 2009.
- [SPE09] SPES. Software Plattform Embedded Systems (SPES2020), [www.spes2020.de](http://www.spes2020.de), 2009.
- [Spt05] UML Profile for Schedulability, Performance and Testing (SPT), 2005.
- [Sys08] SysML Specification 1.1, 2008.
- [Wei08] Tim Weikiens. *Systems Engineering mit SysML/UML*. dpunkt-Verlag, 2008.
- [WW03] Matthias Weber und Joachim Weisbrod. Requirements Engineering in Automotive Development: Experiences and Challenges. *IEEE Software*, 20:16 – 24, 2003.