

# Sicherheit für sicherheitskritische Systeme\*

## Komplexe Systeme mit Model-Checking verifizieren

Mark Brörkens, Ingo Brückner, Ralf Buschermöhle,  
Christoph Schulte und Thomas Wolf

Kuratorium OFFIS, www.offis.de, Tel: 0441-97220

**Zusammenfassung** Mikroprozessoren werden immer häufiger auch in technischen Geräten und vielerlei alltäglichen Konsumgütern eingesetzt. Letzere werden auch als „eingebettete Systeme“ bezeichnet, das heißt in umgebende technische Systeme wechselseitig integrierte Computersysteme. Die Entwicklung solcher „eingebetteter Systeme“ wird unter anderem aufgrund immer höherer Anforderungen an die Funktionalität und einer wachsenden Anzahl von interagierenden Komponenten immer komplexer. Um diesem Umstand Rechnung zu tragen, werden in Prozessmodellen und Standards, die insbesondere im Bereich sicherheitskritischer Systeme angewendet werden, bereits seit geraumer Zeit „vollständige“ Korrektheitsnachweise gefordert. Ein aussichtsreicher Kandidat in diesem Kontext ist das Model-Checking.

## 1 Verifikation komplexer Systeme mit Model-Checking

Mikroprozessoren werden neben ihrer Anwendung in Computern immer häufiger auch in technischen Geräten (wie klinischen Systemen, Kraftfahrzeugen, Telekommunikationsgeräten, industriellen Anlagen) und vielerlei alltäglichen Konsumgütern (wie Fernbedienungen, Waschmaschinen und Küchengeräten) eingesetzt. Letzere werden auch als „eingebettete Systeme“ bezeichnet, das heißt in umgebende technische Systeme wechselseitig integrierte Computersysteme. Die Entwicklung solcher „eingebetteter Systeme“ wird unter anderem aufgrund immer höherer Anforderungen an die Funktionalität und einer wachsenden Anzahl von interagierenden Komponenten immer komplexer [DC01]. Die Beherrschung ihrer Korrektheit mit gängigen Verfahren der Fehlersuche wie Reviews, Simulations- und Testverfahren stößt immer öfter an ihre Grenzen, da diese Verfahren lediglich die manuelle (und damit unter anderem zeit-, kostenintensive und fehleranfällige) Überprüfung einiger Systemläufe zulassen. Zudem sind selbige Zeit- (mittlerweile 70% der Entwicklungszeit) und damit Kostenintensiv und zudem wiederum aufgrund der manuellen Durchführung fehleranfällig.

Um diesem Umstand Rechnung zu tragen, werden in Prozessmodellen (zum Beispiel V-Modell) und Standards (DO-178B), die insbesondere im Bereich sicherheitskritischer Systeme angewendet werden, bereits seit geraumer Zeit „vollständige“ Korrektheitsnachweise gefordert. Ein „vollständiger“ Korrektheitsnachweis betrachtet im Gegensatz zu den herkömmlichen Verfahren immer alle möglichen Systemläufe und wird

\* Dieser Artikel ist in der Zeitschrift „Electronic Embedded Systeme – Elektronik-Magazin für Chip-, Board- u. System-Design“ (AWi Verlag, ISSN 0943-4941, Ausgabe 09/03, September 2003, Seite 19–21) im Schwerpunkt „Embedded-Entwicklungs-Tools“ erschienen.

durch formale Methoden mathematisch bewiesen. Ausgangsbasis für einen solchen Beweis ist eine formale Beschreibung der Anforderungen und der Systemspezifikation. Bislang mussten derartige Nachweise sowie die Formalisierung der Systembeschreibung oftmals manuell von einem Verifikationsexperten angefertigt werden. Dies bringt zwar den Vorteil der Vollständigkeit bei der Systembetrachtung, hat aber prinzipiell die gleichen Nachteile wie bei den bereits erwähnten gängigen Verfahren. Zur Vermeidung dieser Nachteile müssen Automatismen entwickelt werden, die bei der Anfertigung der Beweise unterstützen beziehungsweise diese vollständig rechnergestützt durchführen können.

Ein aussichtsreicher Kandidat in diesem Kontext ist das Model Checking. Hierbei handelt es sich um ein automatisiertes Verfahren zur Verifikation von endlichen, zustandsbasierten, nebenläufigen Systemen. Die Anfertigung des mathematischen Nachweises geschieht vollständig durch den Rechner [CGP99]. Sollte die nachzuweisende Eigenschaft verletzt werden, so kann dem Entwickler (beispielsweise in Form einer computergestützten Debug-Sitzung) automatisch der entsprechende Fehlerpfad aufgezeigt werden.

Das Hauptproblem des Model-Checkings ist das Problem der Zustandsexplosion („state explosion“). Dies begründet sich darin, dass die zu verifizierende Anzahl der Zustände sich exponentiell zur Anzahl der möglichen Wertzuweisungen aller Abläufe des gesamten Systems verhält. Hat das zu verifizierende System viele parallele Komponenten oder Variablen mit großen Wertebereichen, steigt die Anzahl der zu prüfenden Zustände schnell an. Trotz dieser Problematik gelingt es durch aktuelle Forschungsmethoden kontinuierlich die „verifizierbare“ Zustandszahl in beachtlichen Schritten zu vergrößern und so lassen sich bereits jetzt viele Systemklassen erfolgreich automatisch verifizieren.

Da Model-Checker in der Regel spezielle Eingabeformate nutzen, diese jedoch in dieser Form nicht im Entwicklungsprozess auftreten, müssen Transformationen durchgeführt werden, welche die produzierten Artefakte (wie UML-Diagramme, Programmiercode) in ein für den Model-Checker verständliches Format übersetzen. Die Transformationsebenen bieten zudem Einstiegspunkte für die Optimierung des zu verifizierenden Zustandsraums.

Derartige Methoden zur Fehlersuche sind umso interessanter, je früher sie im Entwicklungsprozess eingesetzt werden können, um verbundene Änderungskosten möglichst gering zu halten. In diesem Kontext spielen insbesondere modellgetriebene Entwurfsprozesse eine große Rolle, weil Modelle früh im Entwurfsprozess erstellt werden und einen inhärenten Abstraktionsgrad (Plattformunabhängigkeit) aufweisen sollten. Die große Relevanz derartiger Modelle spiegelt sich unter anderem in den aktuellen Spezifikationsbestrebungen der „Object Management Group“ (OMG) im Rahmen der „Model Driven Architecture“ (MDA) wider.

Bereits 1997 wurde in einem VDI Bericht [DBH<sup>+</sup>01] die Rolle formaler Methoden von BMW herausgestellt. Hierbei wurden Model-Checking Techniken erfolgreich eingesetzt, um Spezifikationsfehler noch vor der ersten Musterphase zu identifizieren und Fehlersuchen durch eine systematische, rechnergestützte Vorgehensweise zu verkürzen. Die Anwendbarkeit von Model-Checking-Techniken konnte mittlerweile in vielen Bereichen der Industrie erfolgreich bestätigt werden (zum Beispiel zur Verifi-

kation komplexer, sequenzieller Schaltungen von verteilten Kommunikationsprotokollen [BDK<sup>+</sup>02]).

Die automatische Transformation der Artefakte im Entwicklungsprozess in Kombination mit der Möglichkeit zur Verifikation vieler praxisrelevanter Systemklassen bereitet die Grundlage, um Model-Checking-Techniken endgültig nahtlos in Entwicklungsprozesse zu integrieren und so die Sicherstellung der Korrektheit von Systemen einen großen Schritt nach vorne zu bringen.

## 2 OFFIS-Verifikationsumgebung

Am Forschungsinstitut OFFIS ist eine Verifikationsumgebung entwickelt worden, die es ermöglicht, Methoden wie formale Verifikation, automatische Testvektorgenerierung und formale Sicherheitsanalyse direkt in Entwicklungswerkzeuge zu integrieren. Neben Statemate wurden zur Erstellung der Modelle unter anderem die Werkzeuge ASCET-SD (ETAS), Rhapsody (I-Logix) und SCADE (Esterel-Technologies) angebunden. Zur Prüfung der Modelle gegen die Spezifikation wird alternativ zum Model Checker VIS [VIS03] zur Zeit der SAT-Solver Prover CL von Prover Technologies eingesetzt. Zur Verdeutlichung der Anwendung von Model-Checking-Techniken bei der Systementwicklung wird im Folgenden ein Verifikationslauf anhand eines kleinen Beispielmodells vorgestellt.

### 2.1 Ein Beispiel: Temperatursteuerung

Für das Beispiel wird das Modellierungswerkzeug ASCET-SD von ETAS verwendet. ASCET-SD ist eine Entwicklungsumgebung für elektronische Steuergeräte wie sie unter anderem in der Automobilindustrie eingesetzt werden. Zur Systemmodellierung bietet die Umgebung sowohl grafische wie auch textuelle Modellierungskonstrukte. Die Verifikation von ASCET-SD-Modellen unter Verwendung der OFFIS-Verifikationsumgebung wird im Folgenden an einem stark vereinfachten Beispiel einer Temperatursteuerung für den Innenraum eines Fahrzeuges dargestellt. Die Temperatursteuerung kennt drei Zustände, „too\_hot“ (zu warm), „too\_cold“ (zu kalt) und „fine\_temp“ (angenehme Temperatur), die in Abhängigkeit von der Temperatur im Fahrzeug angenommen werden. Zusätzlich gibt es einen Initialzustand, der einmalig beim Start des Systems eingenommen wird. Eine akzeptable Temperatur, welche durch den Zustand „fine\_temp“ ausgedrückt wird, liegt in einem Bereich von 20 bis 23 °C vor – abgeleitet davon sind die Temperaturbereiche für „too\_hot“, „too\_cold“ festgelegt. In ASCET-SD ist die Temperatursteuerung in Form eines Zustandsdiagramms (Abbildung 1) modelliert worden, wobei die Zustände durch Rechtecke und mögliche Zustandsübergänge (Transitionen – jeweils mit einer eindeutigen Nummer versehen) durch gerichtete Kanten (Pfeile) visualisiert werden. Sie werden durch Trigger, die in regelmäßigen Intervallen aufgerufen werden, und in Abhängigkeit von Bedingungen, an die sie geknüpft sind, ausgelöst. Die Bedingung, unter der der Trigger die Transition ausführt, steht jeweils in eckigen Klammern hinter dem Wort „trigger“: Sollten mindestens zwei Transitionen schalten können (Nichtdeterminismus), wird diejenige mit der tieferen Nummer gewählt. Die

Fahrzeugtemperatur wird durch die Variable „temp“ beschrieben und durch einen Temperatursensor im Fahrzeuginnenraum gesetzt. Dies geschieht einmal direkt nach der Initialisierungsphase und dann nach jeweils acht Schritten.

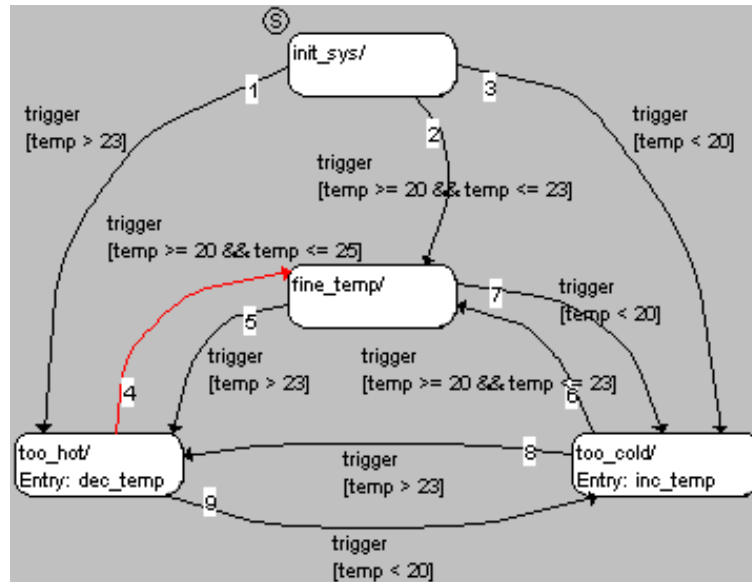


Abbildung 1. Zustandsdiagramm einer vereinfachten Klimasteuerung

## 2.2 Spezifikation der Anforderungen

Zur Modellverifikation ist es erforderlich, die Anforderungen, gegen die das Modell verifiziert werden soll, zu spezifizieren. Die Hauptanforderung ist, dass die Temperatursteuerung nur den Zustand „fine\_temp“ annimmt, wenn die Temperatur im Fahrzeuginnenraum in einem Bereich von 20-23°C liegt (Transitionen: 2, 4, 6). Entsprechend darf der Zustand „too\_hot“ nur beim Überschreiten (größer 23 °C, Transitionen: 1, 5, 8) beziehungsweise beim Unterschreiten (kleiner 20 °C) der Temperatur der Zustand „too\_cold“ (Transitionen: 3, 7, 9) angenommen werden.

Die Formulierung einer solchen Anforderung ist mit Hilfe von vordefinierten Aussage-Mustern („Pattern“) möglich, die die für eine Anwendung oder Domäne typischen Aussagen erfassen. Für dieses Beispiel wird nur der erste Teil der Anforderung spezifiziert und verifiziert. Die Spezifizierung der Anforderung ist mithilfe des Patterns „init\_P\_after\_reaching\_R“ möglich. Ein Pattern stellt eine temporallogische Formel dar. Das vorliegende Pattern steht für die temporallogische Formel  $[\text{not}(R) \text{ Unless } R \text{ and } \odot P]$  und bedeutet, dass der Systemzustand  $P$  initialisiert wird, nachdem  $R$  erreicht wurde.

Pattern bestehen aus einem Start-up-Teil und einem Operationsteil, wobei im Start-up-Teil beschrieben wird, welchen Zustand das System einnehmen soll, bevor es in den gewünschten Zustand übergehen soll. Der nach der Start-up-Phase einzunehmenden Zustand wird dann im Operationsteil beschrieben. Im gewählten Pattern beschreibt R den Zustand, den das System in der Start-up-Phase anzunehmen hat. Entsprechend der Anforderung ist dieses der Zustand „too\_hot“ und eine Temperatur größer 23 °C. Weiter beschreibt P den Systemzustand, den das System nach Ausführung der Start-up-Phase einnehmen soll. Konform zur Anforderung darf nach der Start-up-Phase nicht der Zustand „fine\_temp“ angenommen werden, da die Temperatur noch nicht in dem für „fine\_temp“ erlaubten Bereich von 20 bis 23 °C liegt (siehe Abbildung 2).

```

reporter
Commitments :
=====
fine_temp2 = init_P_after_reaching_R
prop P : not(Module.climate_control::fine_temp);
prop R : (Module.climate_control::too_hot) and (temp > 23);

```

Abbildung 2. Pattern

### 2.3 Ergebnis der Verifikation: Ein Fehlerpfad

Die Ausführung des Pattern-Tests führt zu einem Fehlerpfad (siehe Abbildung 3).

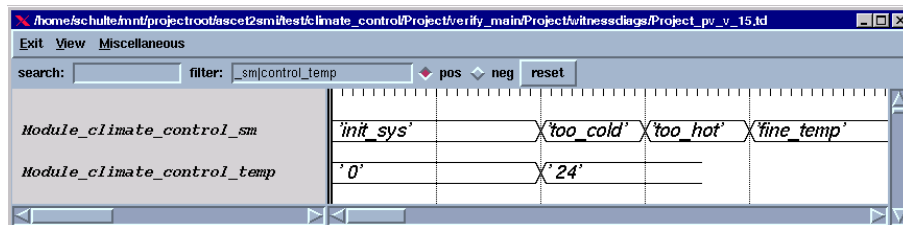


Abbildung 3. Zeitdiagramm

Aus dem Fehlerpfad sind alle Zustands- und Signaländerungen ab dem Systemstart ersichtlich. Das System ist initial im Zustand „init\_sys“. Die Temperatur beträgt zunächst 0°C. Nach der Initialisierungsphase wird, entsprechend der an den Transitionen notierten Bedingungen die Transition vom Zustand „init\_sys“ zum Zustand „too\_cold“ ausgeführt, da die Temperatur unter 20°C liegt. Während des gleichen Schrittes wird vom Temperaturfühler im Fahrzeuginneren eine Temperaturänderung gemeldet. Die

Temperatur beträgt 24°C. Im nächsten Schritt wird daher die Transition vom Zustand „too\_cold“ zum Zustand „too\_hot“ ausgeführt. Die Temperatur ändert sich in den nächsten Schritten nicht mehr. Jedoch wird, nachdem das System den Zustand „too\_hot“ angenommen hat, die Transition zum Zustand „fine\_temp“ aktiviert. Da der Zustand „fine\_temp“ nur angenommen werden darf, wenn die Temperatur kleiner oder gleich 23°C und größer oder gleich 20°C ist, liegt an dieser Stelle ein Modellierungsfehler vor. Die Transition vom Zustand „too\_hot“ zum Zustand „fine\_temp“ (in Abbildung 1 rot markiert) erlaubt entgegen der Anforderung die Ausführung der Transition bei einer Temperatur kleiner oder gleich 25°C.

Die Verifikation des Modells gegen die Anforderung hat ein schnelles und sicheres Auffinden des Modellierungsfehlers ermöglicht. Das Finden von Fehlern wird durch den Einsatz des Model-Checkings auch und gerade bei größeren Modellen erheblich erleichtert und beschleunigt damit den Entwicklungsprozess.

### 3 Praxisrelevanz und Ausblick

Hinsichtlich der Praxisrelevanz haben bereits namhafte Hersteller aus den Bereichen Luft- und Raumfahrt, Automobil- und Bahntechnik die OFFIS-Verifikationsumgebung evaluiert.

Die Evaluationen der Projektpartner zeigten, dass die Formalisierung der Anforderungen in temporallogische Formeln eine nicht zu unterschätzende Aufgabenstellung ist [WDH02]. Da zudem häufig nicht die volle Ausdrucksmächtigkeit von temporalen Logiken benötigt wurde, arbeitet die Abteilung SC (Sicherheitskritische Systeme) bei OFFIS bereits an adäquaten Anwendungsvereinfachungen, wie zum Beispiel den vorgestellten vordefinierten Aussagen-Mustern („Pattern“). Diese Pattern erfassen die für eine Anwendung oder Domäne typischen Aussagen und erlauben einen sicheren Gebrauch. Des Weiteren wird intensiv an weiteren anwendungsfreundlichen, kontextspezifischen Visualisierungen temporallogischer Formeln gearbeitet.

Model-Checking ist bereits seit langem eine interessante Methode zur automatischen Anwendung formaler Verifikationstechniken. Die praktischen Anwendungsgebiete fokussierten bislang jedoch eher auf kleinere Systeme. Aufgrund aktueller Forschungsergebnisse in Kombination mit immer leistungsfähigeren Rechnern konnten diese Grenzen allerdings immer wieder ausgeweitet werden, sodass mittlerweile viele weitere Systemklassen verifizierbar sind. Ist zudem die nahtlose Integration in den Entwicklungsprozess gewährleistet, dürfte es nur noch eine Frage der Zeit sein, bis formale Methoden fester Bestandteil von Entwicklungsprozessen geworden sind.

Im Anwendungsgebiet sicherheitskritischer Systeme ist der Einsatz von Verifikationsmethoden aufgrund der hohen Korrektheitsanforderungen bereits der Normalfall, da ansonsten die kontinuierlich steigende Komplexität (etwa Anforderungen, Entwurfstiefe) dieser Systeme oftmals nicht mehr beherrschbar ist. Model-Checking-Techniken in Kombination mit modellbasierten Entwurfsmethoden garantieren zudem die kostengünstige und flexible Anwendung formaler Verifikationsmethoden, um Fehler im Entwicklungsprozess früh und effizient erkennen zu können. Dies spiegelt sich auch in vielen empirischen Aussagen der Evaluationspartner wider, welche bereits heute die Verifikationsumgebung direkt im Produktivbetrieb einsetzen.

## Literatur

- [BBB<sup>+</sup>99] Tom Bienmüller, Jürgen Bohn, Henning Brinkmann, Udo Brockmeyer, Werner Damm, Hardi Hungar, and Peter Jansen. Verification of automotive control units. *Correct System Design*, 1710:319–341, 1999.
- [BDK<sup>+</sup>02] Jürgen Bohn, Werner Damm, Jochen Klose, Adam Moik, and Hartmut Wittke. Modeling and Validating Train System Applications Using StateMate and Live Sequence Charts. In *Proceedings of the Conference on Integrated Design and Process Technology (IDPT2002)*. Society for Design and Process Science, Society for Design and Process Science, 2002.
- [CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.
- [DBH<sup>+</sup>01] W Damm, U. Brockmeyer, H. J. Holberg, G. Wittich, and M. Eckrich. Einsatz formaler Methoden zur Erhöhung der Sicherheit eingebetteter Systeme im Kfz. *Systemengineering in der Kfz-Entwicklung (VDI Berichte)*, 1374:349–366, 2001.
- [DC01] Werner Damm and Moshe Cohen. Advanced validation techniques meet complexity challenge in embedded software development. *Embedded Systems Journal*, 2001.
- [Nis03] Nissan selects I-Logix' StateMate MAGNUM Tool Chain for Complete Model Based Development Process of Body Electronics Systems. [http://www.ilogix.com/news/press\\_detail.cfm?pressrelease=2002\\_09\\_09\\_115242\\_749386pr.cfm](http://www.ilogix.com/news/press_detail.cfm?pressrelease=2002_09_09_115242_749386pr.cfm), 2003.
- [Saa03] I-Logix' Complete Tool Chain Selected by Saab for Development of Body Electronics Systems. [http://www.ilogix.com/news/press\\_detail.cfm?pressrelease=2003\\_04\\_22\\_110449\\_713180pr.cfm](http://www.ilogix.com/news/press_detail.cfm?pressrelease=2003_04_22_110449_713180pr.cfm), 2003.
- [VIS03] VIS Homepage. <http://www-cad.eecs.berkeley.edu/~vis/>, 2003.
- [WDH02] Hans-Werner Wiesbrock, Heiko Dörr, and Hans Jürgen Holberg. Model Checking im Automotivbereich. In Gerhard H. Schildt and Wolfgang D. Ehrenberger, editors, *Informatik 2002: Sicherheitsrelevante Software in industriellen Anwendungen*. Informatik und Gesellschaft, 2002.