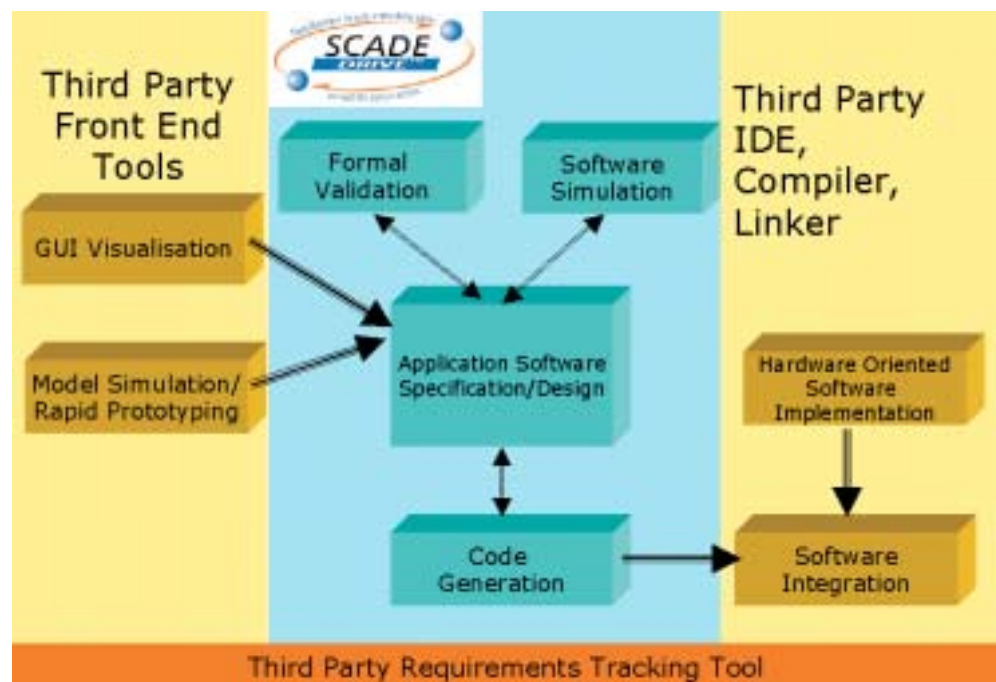


Modellbasierter Entwurf und Formale Verifikation von Embedded-Software

Correct by Construction

15 Jahre nach der Einführung von ‚FlyByWire‘ in der zivilen Luftfahrtindustrie scheint ein ähnlicher Umwälzprozess der Fahrzeugindustrie bevorzustehen. Der Einsatz von softwaregesteuerten elektrischen Systemen in sicherheitsrelevanten Bereichen, wie Lenkung, Bremsen oder Airbag bedingt jedoch eine Abkehr von traditionellen Softwareentwurfsmethoden.

WOLFRAM HOHMANN



Wolfram Hohmann ist seit November 2002 für das weltweite Marketing und Business Development von Esterel Technologies Tools im Automobilmarkt verantwortlich.

Komplexe Embedded-Systeme in Automotiv-Anwendungen müssen absolut fehlerfrei reagieren, da fehlerhaftes Verhalten nicht nur hohe Folgekosten, sondern auch Sach- und Personenschäden nach sich ziehen können. Dies kann nicht durch nachträgliches Testen erreicht werden, die Designprozesse müssen vielmehr dem Anspruch ‚Correct by Construction‘ genügen. Die Erfahrungen der Luftfahrtindustrie haben gezeigt, dass der Einsatz formaler toolgestützter Prozesse nicht nur eine derartige Korrektheit ermöglichen, sondern es im Vergleich zu traditionellen Methoden zu erheblichen Kostenersparnissen kommt.

In dem vorliegenden Artikel wird ein durchgehender Ansatz dargestellt, welcher modellbasierte Softwareentwurfsmethoden für Datenflüsse und Finite Automaten mit den

Möglichkeiten der Formalen Verifikation kombiniert. Des Weiteren wird gezeigt, wie qualifizierte Codegenerierung deutliche Zeitersparnisse mit höchsten Qualitätsansprüchen verbindet. Als Beispiel dient die Toolumgebung ‚Scade Drive‘ von Esterel Technologies. Die Umgebung baut auf Erfahrungen der Luftfahrtindustrie (z. B. Airbus und Eurocopter) mit zertifizierten Arbeitsabläufen auf.

Modellbasierte formale Workflows

Der Entwurf

Konventionelle Workflows – Erstellen einer Spezifikation, manuelles Umsetzen der Spezifikation in ein Softwaredesign, Überprüfen der Umsetzung, manuelles Umsetzen des De-

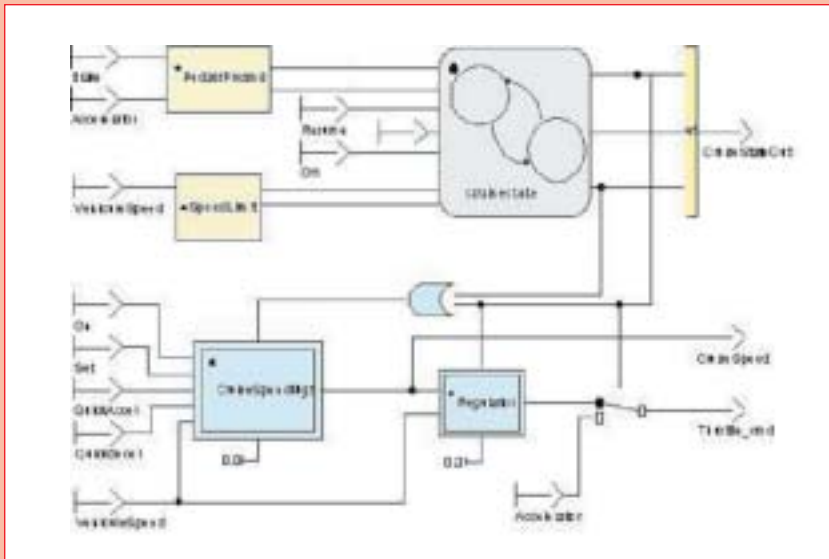


Abb. 1: Graphischen Darstellungen von Datenflüssen

signs in Quellcode, manuelles Generieren von Testfällen und Testen des fertigen Produkts – werden sowohl aus Effizienzgründen als auch hinsichtlich der Fehleranfälligkeit für Embedded-Software-Anwendungen in sicherheitskritischen Bereichen keine Zukunft haben. Hinzu kommt, dass derartige Arbeitsabläufe deutliche Friktionen an den Schnittstellen des Workflows aufwiesen. Diskussionen bezüglich der Erfüllung der Spezifikationen zwischen Auftraggeber und Entwickler sind ebenso unvermeidlich wie Inkompatibilitäten in den unterschiedlichen Wissensbereichen der Beteiligten – die vollständige Umsetzung der Aufgaben erforderten Ingenieurwissen bezüglich der Anwendungen (z.B. in der Regelungstechnik) als auch Programmierkenntnisse bei der Umsetzung. Ein modellbasierter Ansatz vermeidet diese Konflikte. Das Problem wird auf einer abstrakten Ebene graphisch beschrieben. Scade Drive nutzt hierzu zwei Formalismen, die einander ergänzen und zu einer vollständigen Beschreibung des Systemverhaltens auf Anwendungsebene führen.

- ▶ Die Beschreibung von Datenflüssen. Hierbei werden externe und interne Daten mit mathematischen und logischen Funktionen derartig verknüpft, dass hieraus komplexe Systeme entstehen. Die graphischen Darstellungen werden Ingenieuren der Fachrichtungen Regelungstechnik und Digitale Signalverarbeitung bekannt vorkommen (Abb. 1). Auch wenn die Beschreibung der Datenflüsse graphisch erfolgt, liegt dem Modell die synchrone Sprache ‚Lustre‘ zu Grunde. Die Datenflussdiagramme können neben Basisoperationen auch finite Automaten enthalten.
- ▶ Die Beschreibung von Finiten Automaten. Hier wird der Formalismus der ‚SyncCharts‘ angewendet. Die Finiten Automaten erlauben Hierarchien, echten Parallelismus, konditionelle Transitionen und klare, erzwungene Prioritäten (Abb. 2). Im Gegensatz zu anderen Methoden werden Prioritäten eindeutig vom Programmierer erzwungen und nicht vom Tool (zum Beispiel durch Interpretation des Layouts) erzeugt. Grundlegend für die Finiten Automaten ist die Sprache ‚Esterel‘.

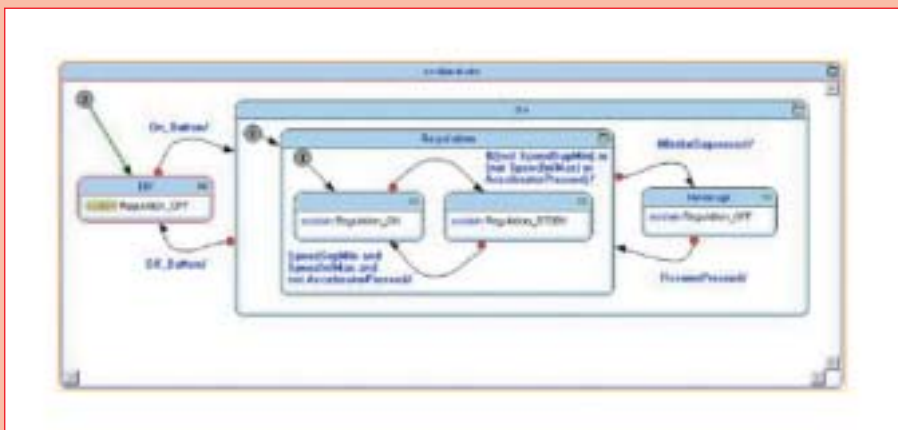


Abb. 2: Die Finiten Automaten erlauben Hierarchien, echten Parallelismus, konditionelle Transitionen und klare, erzwungene Prioritäten

Es ist zu erwähnen, dass beide Methoden synchron und deterministisch sind. Unter synchron ist zu verstehen, dass das zugrundeliegende Verarbeitungsmodell sämtliche existierenden Operationen, Berechnung der Operationen von Datenflüssen sowie Verarbeitung aller Ereignisse in den Finiten Automaten zu bestimmten logischen Zeitpunkten vornimmt. Interne Ereignisse, die durch ‚Feuern‘ der Automaten entstehen, werden nicht über Warteschleifen an das System zurückgeführt, um im nächsten Schritt zur Ausführung zu kommen, vielmehr werden sie im gleichen logischen Zeitpunkt abgearbeitet an dem sie erzeugt wurden. Das gleiche gilt für Datenflüsse, die aufgrund der Bearbeitung mathematischer oder logischer Operationen aus anderen Datenflüssen entstanden sind; alle Operationen erfolgen virtuell zum selben Zeitpunkt.

Die synchrone Abarbeitung bedingt für Datenflüsse, dass ‚Rückkopplungen‘ ohne Einfügung von Operationen zur Zeitverzögerungen verboten sind und durch das Tool als Fehler angemerkt werden. Analoges gilt für Finite Automaten. Zirkelbezüge hinsichtlich der konsumierten und erzeugten Ereignisse zwischen parallelen Automaten werden als Fehlerquelle durch das Tool erkannt und gemeldet. Es ist anzumerken, dass derartige Zirkelbezüge bei nichtsynchrone Systemen, speziell wenn diese auf Warteschleifen aufgebaut sind, potentiell zu Deadlock-Situationen führen. Als Konsequenz haben Kunden, die derartige Tools anwenden, in Designrichtlinien die Benutzung paralleler Automaten unterbunden. Die Verwendung der synchronen Methodik der Toolsuite Scade Drive erfordert keinerlei derartige Einschränkungen. Der Determinismus des Verhaltens leitet sich direkt aus der Synchronität ab. Es wird garantiert, dass bei einem gegebenen Systemzustand eine Folge von Eingabedaten immer die selbe Systemreaktion hervorruft. Hierdurch kann aus dem modellierten und simulierten Verhalten direkt auf das Verhalten des generierten Codes geschlossen werden.

Simulation und formale Verifikation

Wichtige Zielstellung der Einführung formaler modellbasierter Methoden für Embedded-Systeme ist der frühzeitige Nachweis der Korrektheit des Entwurfs. Während im Bereich der digitalen Hardware Methoden wie Simulation und formale Verifikation bereits seit mehreren Jahrzehnten kostenintensive Nacharbeiten minimieren, ist in der Embedded-Software das Testen der fertig programmierten Anwendung integriert auf der Zielhardware nach wie vor Stand der Technik. Die Hauptprobleme die sich hieraus ergeben sind:

- ▶ Das Testen beginnt immer zu spät! Da nach den ersten Tests in der Regel massive Nacharbeiten nötig sind, bedeutet dies, dass ein

Großteil der Softwareentwicklung erst nach der ersten Hardwareintegration erfolgen kann. Die Folge ist, dass ohne eigenes Verschulden die Softwareentwicklung am Ende des Gesamtprojekts den kritischen Pfad darstellt.

- ▶ Die korrekte Analyse der Fehler aufgrund der Einbettung des zu testenden Objektes in das Gesamtsystem ist äußerst schwierig.
- ▶ Die Testüberdeckung kann nicht abgeschätzt werden. Auch monatelange Fahrttests können keinen Ausfall der Software in den ersten Stunden nach Auslieferung des Fahrzeugs oder Monate nach Serienanlauf mit letzter Konsequenz verhindern.

Einer der erfolgversprechendsten Lösungsansätze ist die Simulation. Hier sind zwei grundlegende Voraussetzungen zu garantieren:

- ▶ Das simulierte Verhalten lässt eindeutige Rückschlüsse auf das spätere Systemverhalten zu. Wie wir im vorangegangenen Kapitel gesehen haben ist dieses Problem bei Verwendung synchroner deterministischer Verfahren als gelöst anzusehen.
- ▶ Rapid Prototyping, das heißt, die Simulation des Systemverhaltens an prototypischer Software und die spätere möglichst verhaltensidentische Umsetzung in Produktionscode ist kontraproduktiv, sowohl in Hinblick auf die erreichbare Qualität als auch bezüglich der Produktivität. Qualifizierte Codegeneratoren ermöglichen den Verzicht auf eine Aufteilung des Workflows in Prototypen und Produktion. Lediglich bei der physikalischen Systemsimulation ist eine vorgeschaltete prototypische Entwicklung sinnvoll.

Während die Simulation entscheidend dazu beiträgt, die Entwicklung der Software zu unterstützen, kommt der formalen Verifikation im Qualitätssicherungsprozess eine tragende Rolle zu. Auch wenn es möglich ist, die Testüberdeckung einer Simulation mathematisch zu überprüfen, ist es immer noch schwierig, geeignete Testfälle zu generieren. Hier setzt die formale Verifikation an.

In dem Verfahren des ‚Property Checkings‘ welches bei Scade Drive zum Einsatz kommt, wird mathematisch durch das Tool überprüft, ob bestimmte klar definierte Sicherheitsanforderungen unter allen Bedingungen eingehalten werden. Die Definition der Sicherheitsanforderungen erfolgt Idealerweise ausgehend von FMEAs. Beispiel: eine FMEA fordert, dass in einer Bremsanlage unter keinen Umständen Aus- und Einlassventil gleichzeitig geöffnet werden. Der FMEA-Prozess verlangt in diesen Fällen den eindeutigen Nachweis, was mit Tests und Code Reviews nur schwer zu erreichen ist. Die formale Verifikation ermöglicht es jedoch den Fehlerfall zu modellieren (Ausgang = offen UND Eingang = offen) und mathematisch anhand des existierenden Modells das eindeutige Nicht-Auftreten nachzuweisen. Sollte ein Fehler im Modell gefunden werden, der die Sicherheitsanforderung verletzt, erhält der Entwickler durch detaillierte ‚Counter Examples‘ Hinweise zur Fehlersuche.

Scade Drive bietet formale Verifikation sowohl für finiten Automaten als auch für Datenflüsse an. Bei den letzteren ist es jedoch sinnvoll, den Gültigkeitsbereich von Variablen auf ein vernünftiges Maß einzuschränken; die formale Verifikation könnte unverhältnismäßig lange dauern, wenn z.B. die Einhaltung der Sicherheitsanforderungen für Fahrzeuggeschwindigkeiten zwischen Null und Schallgeschwindigkeit überprüft werden müssten.

Automatische Codegenerierung

Der gesamte Workflow kann nur dann als nahtlos angesehen werden, wenn es gelingt, aus dem erstellten, simulierten und verifizierten Modell automatisch portablen C Code zu generieren. Der Codegenerator muss hierzu bestimmte Voraussetzungen erfüllen:

- ▶ Die Identität des erzeugten Codes mit dem zugrundeliegenden Modell muss eindeutig nachweisbar sein.
- ▶ Der erzeugte Code muss höchsten Qualitätsansprüchen genügen.
- ▶ Der erzeugte Code darf im Vergleich zu

manuell erzeugten Programmen nicht unverhältnismäßig auftragen, sowohl was Codegröße als auch Laufzeit betrifft.

Die Einhaltung des ersten Kriteriums ist dann möglich, wenn es gelingt, den Codegenerator zu validieren. Tatsächlich verfügen z.B. die Esterel-Technologies-Codegeneratoren die ‚DO178 B‘-Validierung für höchste Sicher-

heitsstufen. Eine derartige Anforderung von Seiten der Gesetzgeber existiert zur Zeit für die Automobilindustrie nicht. Es ist anzunehmen, dass es hier bei der gängigen Praxis der Gerichte (insbesondere der US amerikanischen) bleiben wird. den Nachweis des Einhaltens des Standes der Technik zu verlangen. Aufgrund der Diskussionen bezüglich IEC 61508 kann man wohl von der ‚normativen Kraft des Faktischen‘ ausgehen, d.h., dass sich o.g. Norm als Standard durchsetzen wird. Zur Zeit erfolgt die Validierung der Scade-Drive-Toolsuite nach IEC 61508 SIL 4, der höchsten Sicherheitsstufe. Die Qualität von C-Code wird am Besten durch Vergleich mit der MISRA-Norm bewertet. Man sollte hier jedoch anmerken, dass die MISRA-Normen für automatisch generierten Code zum Teil nicht nur sinnlos, sondern sogar kontraproduktiv sind. Der Scade-Drive-Codegenerator hält die MISRA-Normen mit vier dokumentierten und erlaubten Ausnahmen ein.

Die größten Befürchtungen hinsichtlich automatisch generierten Code existieren stets gegenüber der Effizienz. Zahlreiche Benchmarks haben jedoch ergeben, dass zwischen manuell erzeugtem Code und Produkten des Codegenerators Unterschiede nur im statistischen Schwankungsbereich existierten. Dies mag daran liegen, dass modellbasiert entworfene Software von vorneherein Redundanzen und unnötige Programmpfade vermeidet.

Zusammenfassung

Die erhöhten Sicherheitsanforderungen verbunden mit dem steigenden Kostendruck machen es nötig, im Embedded-Software-Bereich für die Automobilindustrie neue Workflows einzuführen.

Der vorliegende Artikel hat gezeigt, wie modellbasierte Arbeitsabläufe und formale Verifikation eingebettet in existierende Prozesse zu deutlichen Verbesserungen der Qualität und der Produktivität führen.

Literatur

- [1] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. Science of Computer Programming, 19(2):87–152, 1992.
- [2] G. Berry. The foundations of Esterel. In G. Plotkin, C. Stirling und M. Tofte, Editoren, Proof, Language and Interaction: Essays in Honour of Robin Milner. MIT Press, 1998.

Beitrag als PDF im Internet:

www.publish-industry.net
more @ click DV63451



Anzeige