Modellbasierte Softwareentwicklung mit SCADE in der Eisenbahnautomatisierung

Stefan Milius, Uwe Steinke

Siemens AG
Industry Sector, Mobility Division, TS RA RD I SWI
Ackerstraße 22
38126 Braunschweig
stefan.milius@siemens.com
uwe.steinke@siemens.com

Zusammenfassung: Wir berichten in diesem Beitrag über ein momentan laufendes Pilotprojekt zur modellbasierten Entwicklung sicherheitsrelevanter Software bei Siemens Mobility im Bereich Eisenbahnautomatisierung. In der Pilotierung benutzen wir SCADE Version 6, ein Tool für die modellbasierte Entwicklung sicherheitsrelevanter Software von der Firma Esterel Technologies. Wir stellen kurz die wesentlichen Merkmale von SCADE vor, und wir berichten von den Erkenntnissen, die wir während unserer Pilotierungsphasen gewonnen haben.

1 Einführung

Die Entwicklung von Produkten für Eisenbahnautomatisierung bringt verschiedene Herausforderungen mit sich. Zunächst einmal müssen die Produkte sehr hohen Sicherheitsanforderungen genügen. Um diese Anforderungen zu erfüllen, müssen aufwändige Tests, Begutachtungen und Zertifizierungen durchgeführt werden. Außerdem haben Produkte der Eisenbahnautomatisierung typischerweise einen Lebenszyklus von über 25 Jahren. Der Wechsel von hardware- zu softwareorientierten Lösungen bringt Schwierigkeiten mit sich: Mit wachsenden Systemanforderungen erhöht sich die Komplexität der Software stetig. Die zugrunde liegende Hardware wird sich im Laufe des Produktlebenszyklus ändern. Von der modellbasierten Softwareentwicklung wird ein Beitrag zur Beherrschung dieser Probleme erwartet. Die Idee ist hierbei, die wesentlichen logischen Anteile der Software eines Systems auf einer Abstraktionsebene zu beschreiben, die weitestgehend unabhängig von der konkret verwendeten Hardwareplattform ist. Das kann bis hin zu einer domänenspezifischen Beschreibung führen. Dabei ist allerdings unabdingbar, dass es keinen Bruch zwischen Modell und Implementierung gibt. Das bedeutet, es muss vermieden werden, dass Modelle und Implementierung nicht mehr konsistent zueinander sind. Dies erfordert den Einsatz von Generatoren zur automatischen Erzeugung von ausführbarem Code aus Modellen. Damit aber die Generatoren für die Entwicklung sicherheitsbezogener Software überhaupt einsatzfähig sind, muss die Qualität der Übersetzung sehr hohen Standards genügen und sie muss gegenüber Zertifizierungsbehörden begründbar sein.

Ein Vorteil der modellbasierten Entwicklung gegenüber klassischer Entwicklung ist die Möglichkeit des Einsatzes von modernen Analysemethoden und -werkzeugen auf Mo-

dellebene (z. B. statische Modellanalyse, Modellsimulatoren oder Model Checking). Dies ermöglicht die frühe Erkennung von logischen Fehlern einer Software, bevor mit speziellen und meist teuren Werkzeugen auf der Zielhardware getestet werden muss. Auch verkürzen sich durch den Einsatz der modellbasierten Entwicklung die Zeiten zwischen Fehlererkennung und Fehlerbehebung.

Für die modellbasierte Entwicklung sicherheitsrelevanter Software nutzt Siemens Mobility in der Rail Automation das Tool SCADE (Certified-Software-Factory) der Firma Esterel Technologies. SCADE erlaubt die Modellierung der logischen Funktionalität von Software eingebetteter Systeme auf einer Abstraktionsebene, die von der unterliegenden Hardwareplattform abstrahiert. Der Modellierungssprache liegt die sogenannte „synchrone Hypothese“ zugrunde: SCADE-Modelle sind zyklisch synchron getaktete Datenfluss- und Zustandsmaschinen. Die SCADE-Modellierungs-Suite unterstützt Simulation und Test auf Modellebene. Ein wesentlicher Bestandteil des Tools ist ein Codegenerator, der für die Softwareentwicklung gemäß den gängigen Normen aus Luftfahrt und Eisenbahnautomatisierung, in naher Zukunft qualifiziert sein wird.

In Abschnitt 2 geben wir einen Überblick über wesentliche Elemente der SCADE-Toolsuite, Version 6. Im Abschnitt 3 berichten wir dann über den Einsatz von SCADE in einem Pilotprojekt bei Siemens Mobility. In Abschnitt 4 diskutieren wir unsere Erkenntnisse aus dem Pilotprojekt. Schließlich geben wir in Abschnitt 5 einen Ausblick auf zukünftige Arbeiten, die modellbasierte Entwicklung mit SCADE bei Siemens betreffen.

2 Überblick über SCADE 6

In diesem Kapitel geben wir einen kurzen Überblick über die wichtigsten Sprachmittel von SCADE 6 und erläutern die wichtigsten Elemente der SCADE-Toolsuite. Grundsätzlich unterscheidet sich die Modellierung in SCADE stark von der Modellierung in bekannten Standard-Modellierungssprachen wie zum Beispiel UML oder SysML. Der Fokus von SCADE liegt in der Beschreibung einer vollständigen Implementierung auf Modellebene und aus unserer Sicht nicht auf der Ebene einer System- oder Softwarearchitekturbeschreibung. Der Abstraktionsgrad der Sprache ist dementsprechend gering, aber angemessen. SCADE bietet eine einfache, mathematisch exakte Semantik, die den Einsatz von Analyseverfahren (statische Modellanalyse oder Model Checking) begünstigt.

2.1 Sprachmittel

Die Modellierungssprache SCADE ist eine Weiterentwicklung der Datenflussbeschreibungssprache LUSTRE [H91].

SCADE liefert Modellierungsmittel für zyklisch synchron getaktete Softwaresysteme. Der gesamten Modellierung in SCADE liegt die „synchrone Hypothese“ zugrunde. Diese besagt, dass die Berechnung eines Taktzyklus eines Modells keine Zeit verbraucht. Diese Annahme stellt natürlich eine Vereinfachung dar, die von realen Systemen nicht erfüllt wird. Dennoch können alle logischen Abläufe und kausalen Zusammenhänge modelliert werden. Für die praktische Anwendung resultiert daraus lediglich die folgende

Restriktion: Die in einem Takt vom Modell verbrauchte Rechenzeit muss kleiner sein als die für einen Takt festgelegte zulässige Zykluszeit.

Hier ein Überblick über die vorhandenen Modellierungsmittel:

Daten: SCADE 6 erlaubt die Behandlung von strukturierten Daten. Alle Datenstrukturen sind statisch. Es ist möglich aus den Grundtypen (real, int, bool) Strukturen und Arrays aufzubauen. Die Modellierungssprache ist stark typisiert.

Datenflussbeschreibungen: Die Modellierungssprache enthält die üblichen logischen und arithmetischen Operatoren sowie konditionale Operatoren. Weiterhin existieren temporale Operatoren für den Zugriff auf Datenflusswerte vergangener Takte. Vorhanden sind die Operatoren für den Zugriff auf Strukturen und Arrays bis hin zu dynamisch indiziertem Array-Zugriff. Es existiert jedoch keine Möglichkeit, explizit Schleifen zu beschreiben. Dafür gibt es die sogenannten „algebraischen Schleifen“ Map und Fold. Diese sind aus der funktionalen Programmierung bekannt. Map dient dazu, einen Operator mit einfachem Datentyp elementweise auf ein Array dieses Datentyps anzuwenden. Mit Fold können Berechnungen über ein Array akkumuliert werden. Map und Fold sind vielseitige Sprachmittel, die bei richtiger Anwendung zu sehr klaren und verständlichen Modellen führen. Allerdings erfordern diese Sprachmittel bei Programmierern aus dem imperativen Umfeld eine „Gewöhnungsphase“. Das Bild 1 zeigt ein Beispiel eines SCADE-Modells.

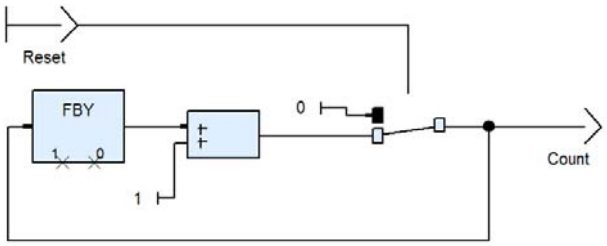

Bild 1: SCADE-Modell eines einfachen Zählers

Zustandsmaschinen: Die Zustandsmaschinen in SCADE sind synchron getaktete, hierarchische Zustandsmaschinen. Sie sind mit den aus der Standard-Modellierungssprache UML bekannten Statecharts vergleichbar. Ein wichtiger Unterschied ist die streng synchrone Semantik: Pro Takt ist in jeder (parallelen) Zustandsmaschine immer nur eine Transition aktiv und wird ausgeführt. Zyklische Abhängigkeiten zwischen Modellelementen sind verboten und werden auch zur Generierzeit wie Syntaxfehler behandelt. Ein Beispiel einer Zustandsmaschine zeigt Bild 2**.**

Ein Vorteil von SCADE 6 ist, dass die beiden Beschreibungsmittel Datenfluss und Zustandsmaschine voll integriert sind und daher beliebig ineinander verschachtelt werden können.

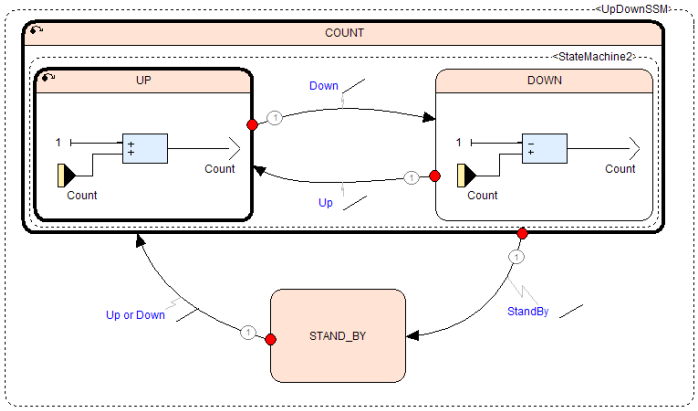

Bild 2: Eine Zustandsmaschine in SCADE

2.2 Tool Suite

Mit der SCADE-Suite steht ein umfangreiches Tool für die Modellierungssprache SCADE zur Verfügung. Bild 3 zeigt die SCADE-Suite im Überblick. Zusätzlich können verschiedene Anbindungen an andere Tools (Gateways) eingesetzt werden:

Grafischer Editor: Jedes syntaktische Modellelement hat eine grafische Repräsentation. Im SCADE-Editor kann mit diesen grafischen Elementen modelliert werden. Es kann auch mit textlicher Modellierungssprache gearbeitet werden.

Codegenerator (KCG): Der Codegenerator erzeugt aus SCADE-Modellen einen C-Code. Die Qualifizierung des Codegenerators für die Entwicklung sicherheitsbezogener Software nach DO 178B, Level A und nach CENELEC EN 50128 für SIL 4, ist für den weiteren Verlauf des Jahres 2008 geplant. Zusammen mit dem bei Siemens Mobility in der Rail Automation eingesetzten zertifizierten CADUL-Compiler ergibt sich eine vollständige Automatisierung der Generierung vom Modell bis ins Zielsytem.

Simulator/Debugger: Er erlaubt, den aus einem SCADE-Modell generierten Code auszuführen, auf Modellebene zu testen und zu debuggen. Der Simulator kann mittels TCL-Skripten gesteuert werden und besitzt eine Automatisierungsschnittstelle.

Model Test Coverage: Diese Toolkomponente erlaubt es, für eine gegebene Testsuite eines Modells strukturelle Abdeckungsmessungen zu machen. Ein Modell kann hierbei für zwei verschiedene Coverage-Kriterien automatisch instrumentiert werden: Decision Coverage (DC) und Modified Condition/Decision Coverage (MC/DC). Andere Kriterien lassen sich auf einfache Weise für (selbstmodellierte) Bibliotheken bereitstellen.

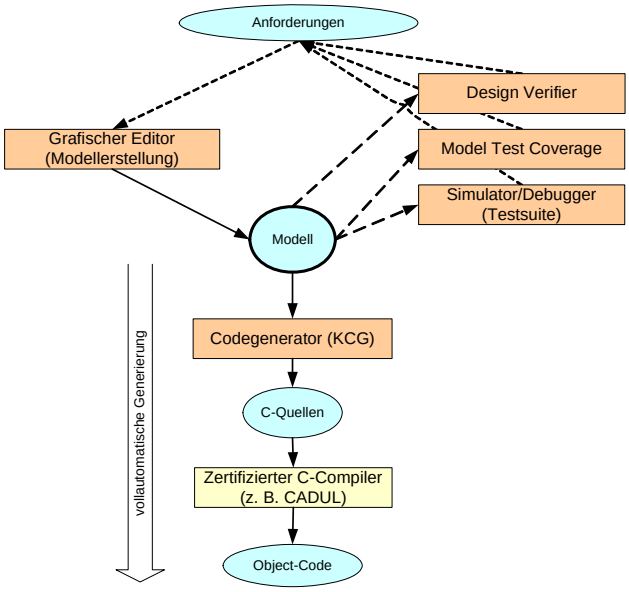

Bild 3: Workflow mit der SCADE-Suite

Design Verifier: Diese Komponente erlaubt die formale Verifikation gewisser Eigenschaften von Modellen. Die zu verifizierenden Eigenschaften werden in SCADE formuliert. Das hat den Vorteil, dass man keine neue Sprache oder Logik zur Spezifikation von Eigenschaften nutzen muss. Allerdings ist die Ausdrucksstärke der Operatoren in SCADE, beispielsweise im Verhältnis zu temporalen Logiken (LTL, CTL), eingeschränkt. Es lassen sich nur sogenannte „Safety-Eigenschaften“ verifizieren, keine „Liveness-Eigenschaften“.[1]

Requirements Gateway: Die SCADE-Suite beinhaltet eine Version des Tools Reqtify (der Firma TNI-Software). Dieses erlaubt eine Verlinkung von Elementen eines SCADE-Modells mit seinen Anforderungen die beispielsweise in einem Tool wie DOORS erfasst sind. Es können auch klassisch kodierte Teile einer Software oder Testfälle verlinkt werden, sodass eine durchgängige toolgestützte Verlinkung von den Anforderungen bis zu Implementation und Test möglich wird.

Weitere Gateways: SCADE besitzt noch weitere Gateways zu anderen Third-Party-Tools: Rhapsody, Simulink und ARTiSAN (das ARTiSAN-Gateway ist für SCADE 6.1, Okt. 2008 angekündigt).

[1] „Safety-Eigenschaften“ sagen aus, dass bestimmte Zustände eines Systems nie eintreten. „Liveness-Eigenschaften“ sagen aus, dass bestimmte erwünschte Zustände in einem Ablauf ausgehend von einem Startzustand irgendwann in der Zukunft sicher auftreten werden.

3 Einsatz von SCADE bei Siemens TS RA

SCADE 6 wird seit Januar 2007 in einem Pilotprojekt bei Siemens Mobility eingesetzt. Bei diesem Pilotprojekt werden Softwarekomponenten von Achszählsystemen mit SCADE modelliert. Achszähler sind Systeme, die in der Bahnautomatisierung zur Gleisfreimeldung eingesetzt werden. Bild 4 zeigt den Einsatz eines Achszählsystems an einem vollautomatischen Bahnübergang.

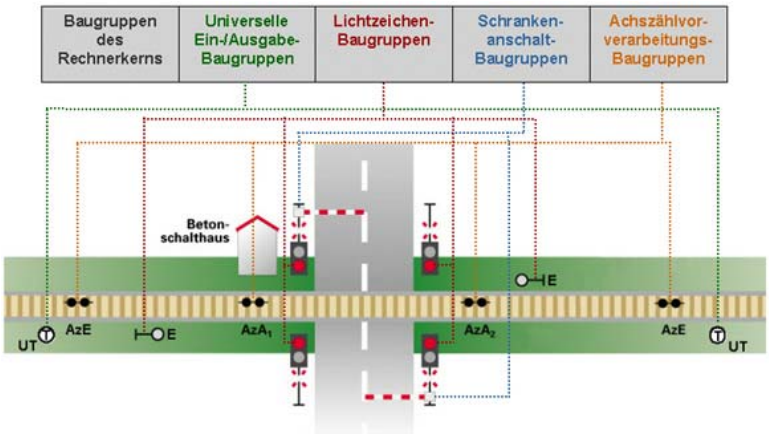

Bild 4: Vollautomatischer Bahnübergang mit Achszählern

Die gezeigten Achszählsensoren (AzE, AzA_1 und AzA_2) beruhen auf dem induktiven Prinzip und werden durch Überfahrung eines Rades beeinflusst, wobei auch die Fahrtrichtung eines Rades ermittelt werden kann. Die Auswertung der Sensordaten findet im Achszählrechner (AzR) statt, der sich im Betonschalthaus befindet. Der AzR ist ein sicherer Rechner nach dem Simis-Prinzip. Der sichere Rechner erfüllt einen Sicherheitsintegritätslevel (SIL) der Stufe 4 – der höchste definierte Level, siehe [N03]. Für die auf dem Rechner laufende Software gilt entsprechend der Software-SIL 4, was eine Reihe von Maßnahmen an den Softwareentwicklungsprozess impliziert, siehe [N01]. Unter anderem werden formale Methoden ausdrücklich empfohlen. Diese Anforderungen liefern eine weitere Motivation für den Einsatz modellbasierter Softwareentwicklungsmethoden.

Im ersten Teil unseres Pilotprojektes wurde in einem bestehenden Achszählsystem der Teil der Software, der die Funktionalität eines Gleisabschnittes trägt, durch ein SCADE-Modell ersetzt. Ziele dieses ersten Teiles des Pilotprojektes waren:

- Schaffung der notwendigen technischen Rahmenbedingungen für den produktiven Einsatz (z. B. Produktionsumgebung, Testumgebung).
- Herstellung der Lauffähigkeit der modellierten Komponenten im realen System.
- Evaluation der Modellierung und der Tool-Unterstützung von SCADE 6.
- Erfahrungsaufbau mit der SCADE-Methode.

Nach Abschluss der ersten Phase des Pilotprojektes und den dabei gewonnenen Ergebnissen wurde beschlossen, SCADE auch im produktiven Bereich einzusetzen. Seit September 2007 wird nun in der zweiten Phase der SCADE-Pilotierung eine Komponente, eines in der Entwicklung befindlichen Systems, mit SCADE modelliert. Es handelt sich um ein Achszählsystem, das in Stellwerken zur Gleisfreimeldung eingesetzt wird. Die wichtigsten Ziele der zweiten Phase sind:

- Durchgängige Entwicklung des Achszählsystems von den Anforderungen bis zur Zertifizierung.
- Einsatz von formaler Verifikation zur frühen Erkennung von Fehlern.
- Einsatz weiterer modellbasierter Techniken, um SCADE in einen durchgängigen modellbasierten Entwicklungsprozess zu integrieren.

Es wurde eine Software-Komponente mit SCADE modelliert, die das Grundstellverfahren eines Gleisabschnittes in einem Stellwerk implementiert. Durch die Bedienhandlung des Grundstellens wird ein Belegungszustandswechsel eines Gleisabschnittes von „besetzt“ nach „frei“ erzwungen. Die Bedienhandlung ist in einem Stellwerk nur in Sonderfällen zulässig und an Bedingungen geknüpft. Das Grundstellverfahren überwacht die Einhaltung dieser Bedingungen und führt den Grundstellvorgang aus.

Die Anforderungen an die Software-Komponente wurden in dem Requirements-Engineering-Tool DOORS [D] erfasst und mit der Umsetzung in SCADE verlinkt. Auf diese Weise ist die Nachverfolgbarkeit besser gegeben, als durch die Norm [N01] vorgeschrieben In der Norm ist keine Verlinkung bis auf Quellcodeebene (also in unserem Fall bis auf SCADE-Modell-Ebene) gefordert – das ist allerdings durch unser Vorgehen sicher gestellt.

Im Rahmen der Pilotierung wird von uns ein statistisches Testmodell erstellt. Dies wird unabhängig von dem Implementationsmodell in SCADE aus den Anforderungen abgeleitet. Bild 5 zeigt das grundsätzliche Vorgehen, siehe auch [Sc07]. Bei dem Testmodell handelt es sich im Wesentlichen um eine Markov-Kette. In einer Kooperation mit dem Fraunhofer IESE setzen wir im Rahmen einer Diplomarbeit, die am Institut für Software und Systems Engineering der TU Braunschweig angesiedelt ist, das Tool JUMBL [J] ein, um aus diesem Testmodell Testfälle zu generieren. Diese können für Belastungstests des Modells benutzt werden. Mit Hilfe der Wahrscheinlichkeiten in der Markov-Kette können bestimmte Anwendungsszenarien häufiger oder weniger häufig getestet werden. Es hat sich gezeigt, dass durch dieses Vorgehen Fehler schon in der Modellierungsphase erkannt werden, die ansonsten erst in viel späteren Phasen entdeckt würden.

Im Rahmen des Testens des SCADE-Modells wird auch von dem Model-Test-Coverage-Feature der SCADE-Tools Suite Gebrauch gemacht. Wie bereits erwähnt lässt sich damit die strukturelle Abdeckung des Modells durch die vom Testmodell generierte Testsuite nachweisen. Dies ist auch eine Forderung aus der Norm [N01] für den Software-SIL 4.

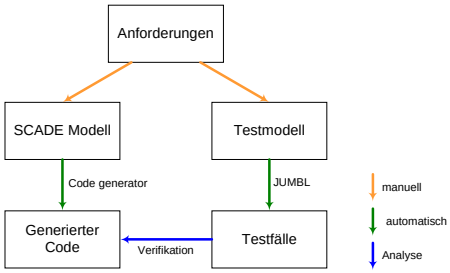

Bild 5: SCADE-Modell und Testmodell

Ein Problem beim Test ergibt sich durch die Einbettung des Modells in das Zielsystem oder in klassisch kodierte Software. Um diese Einbettung zu erreichen, muss ein Wrapper implementiert werden, der die Schnittstellen des Modells in die des schon vorhandenen Ziel-Softwaresystems umsetzt. Da dieser Wrapper klassisch in der Zielsprache programmiert wird, lässt er sich nicht mehr auf Modellebene testen. Hier ist ein Modul- und Integrationstest nötig. Dieser Test kann innerhalb einer vorhandenen Testumgebung für Softwarekomponenten geschehen. In der Pilotierung wird für diesen integrativen Testschritt das Tool RT-Tester von Verified International [V] eingesetzt.

4 Ergebnisse und Erfahrungen

In diesem Abschnitt berichten wir von den Erkenntnissen, die wir im Laufe der zwei Pilotierungsphasen gewonnen haben.

Ein wesentliches Ziel des Einsatzes eines modellbasierten Softwareentwicklungstools war die Trennung von funktional logischen Anteilen der Software von der spezifischen Hardwareplattform. Dieses kann mit SCADE sehr gut umgesetzt werden. Die gewählte Abstraktionsebene erlaubt und erzwingt aufgrund ihrer präzisen Semantik einerseits die vollständige Beschreibung der logischen Funktionalität und abstrahiert andererseits hinreichend vom Zielsystem. Durch die Verwendung des Codegenerators geht das Modell ohne Bruch in die Implementierung ein. Dadurch sind unserer Meinung nach SCADE-Modelle auch prinzipiell geeignet, leicht auf andere Zielsysteme übertragen zu werden. Dies kann durch Änderung der dünnen Wrapper-Schicht eines Modells geschehen oder durch Verwendung eines anderen Code-Generators. Die Übertragung auf andere Zielsysteme ist aber bei unserer Pilotierung noch nicht verifiziert worden. Insgesamt sind das SCADE zugrunde liegende Paradigma eines zyklisch getakteten Systems und die synchrone Hypothese gut für die Umsetzung typischer Automatisierungsaufgaben geeignet. So arbeiten beispielsweise typische Prozessrechner als zyklisch getaktete Systeme.

Bei dem vorhandenen Codegenerator gibt es verschiedene Optimierungsstufen, wobei der Code, der auf einer niedrigen Optimierungsstufe generiert wird, eine noch unzureichende Qualität hat. Code der höchsten Optimierungsstufe 3 hat eine brauchbare Qualität. Modellweite globale Optimierungen waren bei unseren Beispielen im generierten Code nicht erkennbar. Eine Schwierigkeit ist, dass in der Schnittstelle des zu einem Modell generierten Codes die Trennung von internem Zustand und Ein-/Ausgabedaten

nicht durchgängig ist: Der Codegenerator generiert zu jedem SCADE-Knoten nur eine C-Datenstruktur, die sowohl die Ausgabedaten als auch die Daten des internen Zustandes des SCADE-Knotens enthält. Dies beeinträchtigt die Austauschbarkeit von Modellen innerhalb ihrer Ablaufumgebung erheblich. Hierbei sollte ein Modell durch ein abweichend implementiertes Modell mit derselben Schnittstelle einfach zu ersetzen sein. Außerdem entspricht der generierte Code nicht den MISRA-Richtlinien [M]. Der C-Code, der mit der Optimierungstufe 0 aus hierarchischen Zustandsmaschinen generiert wird, beinhaltet an mehreren Stellen ineinander verschachtelte Switch-case-Blöcke. Diese haben nur eine Auswahlalternative und besitzen keinen Default-Block.

Es gibt einige problematische Punkte bei der Modellierung. Zunächst ist durch die relativ niedrige Abstraktion eine Modellierung auf der Ebene einer Systemarchitektur nicht möglich. Die Modellierung mit SCADE ist eher auf der Ebene der Implementierung angesiedelt. Es ergibt sich somit momentan noch eine Modellierungslücke zwischen den Anforderungen und der Implementierung. Gegenwärtig wird von Esterel Technologies ein Gateway zwischen SCADE und ARTiSAN entwickelt, der diese Lücke schließen soll.

Eine weitere Schwierigkeit ist, SCADE-Modelle in Umgebungen zu integrieren, die nicht dem zyklisch synchronen Paradigma entsprechen. Dieser Anwendungsfall tritt auf, wenn, wie in unserem Fall, nur Teile einer „Alt-Software“ durch SCADE-Modelle ersetzt werden oder wenn das Zielsystem selbst keine zyklische synchrone Datenflussmaschine darstellt. Zwar ist die Einbettung in solchen Fällen trotzdem mit einem geeigneten Wrapper machbar, die Schnittstelle zwischen Modell und Laufzeitumgebung muss dann aber zwischen beiden „Welten“ umsetzten. Das führt dazu, dass Ergebnisse, die mittels formaler Verifikation auf Modellebene gewonnen wurden, einer sorgfältigen Interpretation bedürfen. Zudem muss in einem solchen Fall der Wrapper zusätzliche Funktionalitäten übernehmen. Er wird damit so komplex, dass ein eigener umfangreicher Modultest erforderlich ist.

Beim Thema Test sind unsere Erfahrungen positiv. Durch die Beschreibung der logisch funktionalen Anteile einer Software auf Modellebene können diese Anteile frühzeitig unabhängig von der Zielhardware verifiziert werden. Durch die synchrone Hypothese gibt es jedoch wichtige Aspekte, die nicht ohne Weiteres im Modell getestet werden können – das sind Echtzeitaspekte und Performance. Diese müssen weiterhin auf der Zielhardware getestet werden. Ein Schwachpunkt beim Thema Test ist die fehlende Integration des SCADE-Simulators mit externen Debuggern. Eine Testumgebung, in der man SCADE-Modelle gemeinsam mit handgeschriebenem Code testen kann, existiert leider noch nicht.

Begleitend zu unserer Pilotierung von SCADE hat eine unabhängige Evaluation der SCADE-Suite durch Siemens CT stattgefunden. Die Ergebnisse dieser Evaluation sind positiv ausgefallen. Bei der Untersuchung wurde ein Entwicklungsprozess, der auf SCADE basiert, mit einem ausgereiften Prozess verglichen, der die Programmiersprache C benutzt. Es wurde festgestellt, dass SCADE insgesamt positiv gegenüber der konventionellen Entwicklung abschneidet, wobei allerdings die Stärken von SCADE auf Konstruktion und Design liegen. Bei der Testunterstützung wurde noch Verbesserungsbedarf gesehen.

5 Fazit und Ausblick

Zusammenfassend lässt sich feststellen, dass SCADE 6 trotz Einschränkungen bereits eine modellbasierte Entwicklung realisiert. Die Trennung von Logik und Hardware wird erreicht, sodass gerade bei langen Produktlebenszyklen der wesentliche Inhalt eines Systems unabhängig von der Hardware erhalten bleibt. Auch scheint der Einsatz von Analysetechniken vielversprechend (z. B. Model Checking mit dem SCADE Design Verifier). Auf diesem Gebiet konnten wir aber leider aufgrund der fehlenden Verfügbarkeit des Design Verifiers für SCADE 6 noch keine Erfahrungen sammeln. Ob der Einsatz von Model Checking nur zur Testunterstützung dient oder letztlich die Zertifizierung (teil)modellierter Systeme vereinfacht, bleibt im Moment noch offen. Hier sind noch intensive Gespräche mit den Zertifizierungsbehörden erforderlich.

Bis ein breiter Einsatz von SCADE im produktiven Bereich erfolgen kann, müssen einige Verbesserungen realisiert werden – eine Toolstabilität ist noch nicht sichergestellt. In Zukunft wäre auch ein besseres Konzept zur Integration von Testumgebungen für Modelle mit solchen für klassisch programmierte Software und eine intuitivere Benutzerführung im Editor wünschenswert. Wir werden im Rahmen unsere Pilotierung untersuchen, wie sich Testfälle aus den Tests der Modellebene für den Integrationstest, der den Wrapper beinhaltet, wiederverwenden lassen.

Trotz Einschränkungen werden bei Siemens Mobility in der Rail Automation die Aktivitäten zur modellbasierten Entwicklung mit SCADE intensiviert. SCADE wird beispielsweise für die Softwareentwicklung von Zugbeeinflussungssystemen eingesetzt. Hierbei wird auf den relevanten Zielrechnern der Simis-Reihe eine Ablaufumgebung eingesetzt werden, die sich an dem Execution-Chain-Pattern [Sz06] orientiert und zyklisch getaktet arbeitet. Diese Ablaufumgebung bietet SCADE-Modellen einen geeigneten Rahmen.

Literaturverzeichnis

[D] Telelogic DOORS, Telelogic AB, Malmö, http://www.telelogic.com/products/doors

[J] Java Usage Model Builder Library (JUMBL), Software Quality Research Laboratory, Univesity of Tennessee, http://www.cs.utk.edu/sqrl/esp/jumbl.html

[H91] Halbwach, N. et al: The synchronous dataflow programming language LUSTRE, *Proceedings of the IEEE* 79, Nr. 9, 1991, 1305-1320.

[M04] MISRA: MISRA-C: Guidelines for the Use of the C Language in Critical Systems, Oktober 2004.

[N01] Norm CENELEC EN 50128: Railway application – Communications, signaling and processing systems – Software for railway control and protection systems, März 2001.

[N03] Norm CENELEC EN 50129: Railway application – Communications, signaling and processing systems – Safety related electronic systems for signaling, Dezember 2003.

[Sc07] Schieferdecker, I.: Modellbasiertes Testen, *OBJEKTspektrum* Mai/Juni 2007, 39-45.

[Sz06] Schütz, D.: Execution Chain. In Longshaw, A.; Zdun, U.(Hg.): Proceedings of the 10th European Conference on Pattern Languages of Programs 2005, 1. Auflage 2006.

[V] Verified Systems International GmbH, Bremen, http://www.verified.de