Vol. 13
Ausgabe 1.08

eclipse MAGAZIN

eclipse MAGAZIN

www.eclipse-magazin.de

S&S

Mit CD
Alle Infos S. 3

Tools, Open Source & more

- Eclipse Visual Editor Project
- Apache Tomcat 6.0
- ATLAS Transformation Language
- Eclipse Modeling Framework (EMF)
- Eclipse Graphical Modeling Framework (GMF)
- Graphical Editing Framework (GEF)
- Standard Widget Toolkit (SWT)

BONUS:
Artikelserie aus dem Java Magazin „OSGi applied“
+ Bonusartikel aus dem Eclipse Magazin Vol. 12

Professionell Testen

Best Practices: GUI-Tests für Eclipse RCP

Embedded Eclipse:
Model Driven

itemis
Software | Consulting | Coaching

Sonderdruck der Firma itemis

Rich Clients
Neu seit Eclipse 3.3:
Eclipse DataBinding

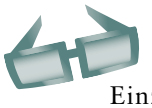
Model Driven
Dokumentationen generieren

Plattformen
Eclipse und NetBeans Platform im Vergleich

Modellgetriebene Softwareentwicklung

MDA und MDSD in der Praxis

GEORG PIETREK, JENS TROMPETER, JUAN CARLOS FLORES BELTRAN, BORIS HOLZER, THORSTEN KAMANN, MICHAEL KLOSS, STEFFEN A. MORK, BENEDIKT NIEHUES, KARSTEN THOMS



Die modellgetriebene Entwicklung findet immer mehr Einzug in das Standardrepertoire der Softwareentwicklung. Auch die Anzahl der Bücher zum Thema MDA/MDSD nimmt nun deutlich zu. Im Gegensatz zu vielen theoretischen Abhandlungen wird im Buch „Modellgetriebene Softwareentwicklung" das Thema aus der Sicht der Praxis betrachtet, ohne sich dabei auf eine spezielle Technologie zu beschränken. Nichtsdestotrotz findet sich in diesem Buch ein deutlicher Bezug zu Eclipse-Technologien.

Die insgesamt neun Autoren des Buchs sind alle bei der itemis AG in Lünen beschäftigt, die seit 2003 Software, Services und Beratung rund um Methoden und MDA/MDSD anbietet und auch die Weiterentwicklung des MDA-Werkzeugs openArchitectureWare unterstützt, das seit Version 4.0 zum Eclipse-GMT-Projekt (Generative Modeling Technologies) gehört. Die langjährige Erfahrung der Autoren mit modellgetriebener Entwicklung in der Praxis und der Einführung von MDA/MDSD in verschiedenen Unternehmen ist deutlich spürbar. Während viele Bücher bei der Entscheidung, ob modellgetriebene Entwicklung eingesetzt werden sollte, wertvolle Hintergrundinformationen liefern können, geht dieses Buch darüber hinaus: Ist die Entscheidung für MDSD bereits gefallen, kann dieses Buch fundierte Antworten darauf bieten, was für ein Tool eingesetzt werden sollte und wie.

In den insgesamt sieben Kapiteln richtet sich das Buch an nahezu alle Beteiligten der Softwareentwicklung. Von IT-Verantwortlichen, Projektleitern und Analysten über Softwarearchitekten und -entwickler bis hin zu Test- und Build-Managern. In der Einleitung werden die grundlegenden Begriffe geklärt und gezeigt, warum MDA/MDSD eingesetzt werden sollte.

Die wichtige Frage, wie MDSD in verschiedene etablierte Entwicklungsprozesse integriert werden kann, wird in Kapitel 2 am Beispiel des V-Modells XT, RUP und agile Entwicklung diskutiert, wobei deutlich die große Erfahrung der Autoren mit verschiedenen Firmen, verschiedenen Prozessen und den realen Gegebenheiten zum Tragen kommt. Kapitel drei widmet sich verschiedenen Aspekten der Modellierung, wie etwa der Metamodellierung, der domänenspezifischen Modellierung sowie der Modellvalidierung und -transformation, und gibt eine gute Übersicht über verschiedene Werkzeuge für MDA/MDSD mit dem Fokus auf Open-Source-Tools. Die Anwendung domänenspezifischer Sprachen (DSL) in der Praxis wird in Kapitel 4 anhand von WebML und der Entwicklung einer DSL für Tests mit FIT und FitNesse erläutert. Besonders hilfreich ist die Vorstellung von Best Practices zur modellgetriebenen Entwicklung in Kapitel 5. Dabei wird u.a. die Entwicklung eigener Cartridges, die Integration in den Build-Prozess, die Generierung von Dokumentation und der Umgang mit generiertem und handgeschriebenem Code thematisiert. In Kapitel 6 wird anhand eines ausführlichen Anwendungsbeispiels die modellgetriebene Entwicklung einer Drei-Schicht-Architektur auf Basis von EJB3 und JSF vorgestellt. Als Modell dient ein UML2-Modell mit Stereotypes und als MDSD-Framework findet dabei openArchitectureWare Verwendung. Dabei wird insbesondere die Entwicklung der unterschiedlichen Templates im Detail diskutiert, was einen guten Einstiegspunkt für eigene Entwicklungen bietet.

Auf der CD zum Buch findet sich die komplette Software für das Anwendungsbeispiel inklusive Eclipse 3.2.1, JBoss, JDK 1.5, openArchitectureWare 4.1.2 und die drei Eclipse-Projekte für das Anwendungsbeispiel mit einem kleinen Schönheitsfehler: Anstelle einzelner Dateien, findet sich auf der CD ein CD-Image. Um dieses Problem zu beheben, bietet der Verlag den unproblematischen Versand einer Ersatz-CD an. Wer es etwas eiliger hat, der kann das Image auf die Festplatte kopieren und daraus dann eine neue CD brennen.

Schade ist, dass die Grafiken im Buch nicht wirklich gelungen sind. Sie wirken einerseits sehr inkonsistent und ferner lässt die Qualität einzelner Grafiken zu wünschen übrig. Bei der Produktion der ersten Auflage des Buches gibt es leider noch einen weiteren Mangel: Das Literaturverzeichnis fehlt, kann aber auf der Website zum Buch als PDF heruntergeladen werden. Diese kleinen Kinderkrankheiten können jedoch den insgesamt sehr guten Eindruck dieses Buchs nicht trüben. Wer ernsthaft modellgetriebene Entwicklung einsetzen will, dem ist es wärmstens zu empfehlen. Trotz der großen Anzahl von Autoren ist der Schreibstil recht flüssig und liefert vor allem zur Frage, wie MDSD eingesetzt werden soll, sehr fundierte Informationen. Ferner kann das Buch für die Auswahl eines passenden Tools aus dem wachsenden Dschungel von Open-Source-Tools für modellgetriebene Entwicklung als Wegweiser dienen.

Alexander Schwartz

Modellgetriebene Softwareentwicklung

MDA und MDSD in der Praxis

Georg Pietrek, Jens Trompeter, Juan Carlos Flores Beltran, Boris Holzer, Thorsten Kamann, Michael Kloss, Steffen A. Mork, Benedikt Niehues, Karsten Thoms

256 Seiten, 39,90 Euro

entwickler.press 2007

ISBN 978-3-939084-11-2

MDSD für eingebettete Systeme mit Eclipse – die Herausforderungen

Welten wachsen zusammen

>> WOLFGANG NEUHAUS, BENEDIKT NIEHUES UND LOTHAR WENDEHALS

Entwicklungsmethoden und -werkzeuge für eingebettete Systeme und für Enterprise-Anwendungen nähern sich zusehends an. Doch neben Faktoren, die für eine zunehmende Annäherung sprechen, müssen Besonderheiten bei der Entwicklung eingebetteter Systeme berücksichtigt werden, die es bei Enterprise-Anwendungen nicht gibt oder die nicht so ausgeprägt sind. Diese wollen wir im Folgenden betrachten.

Die Werkzeuglandschaft im Entwicklungsprozess für eingebettete Systeme ist breit gefächert und die Werkzeuge reichen von Speziallösungen bis hin zu universell einsetzbaren Entwicklungsumgebungen. Eine weit verbreitete Entwicklungsumgebung ist zum Beispiel MatLAB/Simulink [1]. Diese Umgebung bietet modellbasiertes Design und Simulation dynamischer Abläufe wie Regelungssysteme. Bei der Modellierung ganzer Systeme sind Entwicklungsumgebungen auf Basis der UML und SysML wie Telelogic Rhapsody [2] im Einsatz. Spezielle Entwicklungsumgebungen wie die ASCET-Werkzeuge [3] werden in der Steuergeräteentwicklung für den Automobilbau verwendet.

Werden unterschiedliche Werkzeuge innerhalb eines Entwicklungsprozesses in Kombination eingesetzt, müssen die Modelle problemlos zwischen den Werkzeugen austauschbar sein. Dann kann für einen bestimmten Zweck jeweils das Werkzeug zum Einsatz kommen, das am besten geeignet ist, sodass der Entwicklungsprozess optimiert werden kann. Andernfalls drohen zum Beispiel Modelle verschiedener Werkzeuge untereinander inkonsistent zu werden und damit hohe Kosten und im Bereich sicherheitskritischer Systeme sogar hohe Risiken für Menschen zu verursachen. Die modellbasierte Softwareentwicklung kann hier helfen, den Entwicklungsprozess zu optimieren und Kosten sowie Risiken zu senken. Die Methoden der modellbasierten Entwicklung helfen

sowohl bei der Integration der Modelle verschiedener Werkzeuge als auch bei der Vereinfachung und Automatisierung der Prozessschritte in der Entwicklung. Dabei gibt es im Eclipse-Umfeld eine Reihe von fertigen Werkzeugen, die dafür eingesetzt werden können

Werkzeugintegration

Zur Integration von Modellen verschiedener Werkzeuge sind grundsätzlich zwei Vorgehensweisen möglich: ein gemeinsames Modell für alle Werkzeuge oder die Synchronisation der einzelnen Modelle. Bei einem gemeinsamen Modell wird das spezialisierte Modell eines Werkzeugs durch Modell-zu-Modell-Transformation aus dem gemeinsamen Modell gewonnen. Nach der Bearbeitung mit dem Werkzeug werden die Änderungen des spezialisierten Modells zurück in das gemeinsame Modell transformiert. Bei der Synchronisation einzelner Modelle werden dagegen Änderungen zwischen jeweils zwei Modellen direkt von einem in das andere Modell propagiert. Eventuelle Transformationen werden bei der Propagierung durchgeführt. Voraussetzung für die Integration sind allerdings offene Programmierschnittstellen, die den Zugriff auf die Modelle der verschiedenen Werkzeuge ermöglichen. Solche Integrationen sind allerdings häufig Speziallösungen, die nicht von den Werkzeugherstellern angeboten werden, sondern von Drittanbietern maßgeschneidert werden müssen. Modell-zu-Modell-Transformationen können im Rahmen von Eclipse beispielsweise mit Xtend aus dem openArchitectureWare-Projekt [4] oder mit der ATLAS Transformation Language [5] durchgeführt werden.

Metamodellierung, DSLs und Editoren

Betrachtet man die derzeit bei der Entwicklung von eingebetteten Systemen verwendeten Modellierungssprachen, so wird das „Zusammenwachsen der Welten“ besonders deutlich. Waren vormals Blockschaltdiagramme, die Architecture Analysis and Design Language (AADL), Message Sequence Charts (MSC) oder die Specification and Description Language (SDL) vorherrschend, so werden

> Es werden zunehmend DSLs und spezialisierte Sprachen eingesetzt.

in der Entwicklung zunehmend UML-basierte Domain Specific Languages (DSL) bis hin zu spezialisierten Sprachen zum ganzheitlichen Systems Engineering wie SysML [6] eingesetzt. Durch entsprechende UML-Profile wie die für SysML, AUTOSAR oder MARTE lässt sich die allgemein verwendbare UML2 bis zu einem gewissen Grad spezialisieren. In eigenen, daraus abgeleiteten Profilen lässt sich dies für eigene Zwecke weiterführen.

Doch auch textuelle DSLs finden häufig Verwendung, weil viele Modelle für eingebettete Systeme derzeit auf einem plattformnahen, niedrigen Abstraktionsniveau beschrieben werden. Dies erhöht zwangsläufig den Detaillierungsgrad und damit die Komplexität. Textuelle DSLs eignen sich besonders gut zur Erfassung vieler Details. Grafische DSLs hingegen eignen sich eher zur Visualisierung von Strukturen, Beziehungen oder Abläufen. Optimal wäre es also, vorhandene DSLs mit den dazugehörigen Metamodellen und Editoren mit eigenen DSLs zu kombinieren und für jeden Stakeholder die am besten geeignete Sicht (textuell/grafisch) bereitzustellen. Eclipse bietet die dazu notwendigen Basiswerkzeuge im Eclipse-Modeling-Projekt [7]. Das Eclipse Modeling Framework (EMF) [8] hat sich als Standard für Metamodellierung und -modelle etabliert und wird von vorhandenen Editoren unterstützt. Mithilfe des Graphical Modeling Framework (GMF) [9] können spezielle grafische DSLs umgesetzt werden. Aus dem TOPCASED-Projekt [10] stammt die Alternative Topcased MF für grafische DSLs. Xtext aus dem openArchitectureWare-Projekt [4] ermöglicht die einfache Erstellung textueller DSLs mit dazu gehörigen Editoren und Parsern auf Basis von Grammatiken. Xtend, ebenfalls Teil des openArchitectureWare-Projekts, hilft dabei, Metamodelle nicht invasiv zu erweitern und Modelltransformationen zu beschreiben. Dies ist besonders bei der Verbindung von grafischen und textuellen DSLs hilfreich. Eclipse hilft also dabei, einen geeigneten Mix aus Modellierungssprachen und -editoren herzustellen, der die eigene Domäne optimal unterstützt. Eigene Integrationsarbeit ist allerdings notwendig.

Validierung und Verifikation

Die sehr detaillierten und umfangreichen Modelle eingebetteter Systeme erfordern eine frühe, automatisierte Überprüfung der Korrektheit, damit Auswirkungen von Änderungen nachvollziehbar bleiben und unnötige Build-/Deploy-/Test-Zyklen vermieden werden. Das dem Modell zu Grunde liegende Metamodell kann jedoch nur teilweise die syntaktische Korrektheit des Modells sicherstellen. Auch Einschränkungen bei der Freiheit der Modelleditoren sind nur bis zu einem gewissen Grad praktikabel. Einige syntaktische, aber vor allem semantische Vorgaben an das Modell, können weder vom Metamodell noch von speziellen Editoren gewährleistet werden. Deshalb ist eine explizite Validierung des Modells durch Regeln notwendig.

Die Regelsätze, die zu einer umfassenden Überprüfung des Modells benötigt werden, sind bei der Entwicklung eingebetteter Systeme häufig sehr groß. In der Praxis sind Regelsätze von bis zu 1 000 Regeln nicht ungewöhnlich. Je komplexer und detaillierter die Modelle werden, umso umfangreicher werden auch die Regelsätze. Die Größe der Regelsätze hat entscheidenden Einfluss auf die

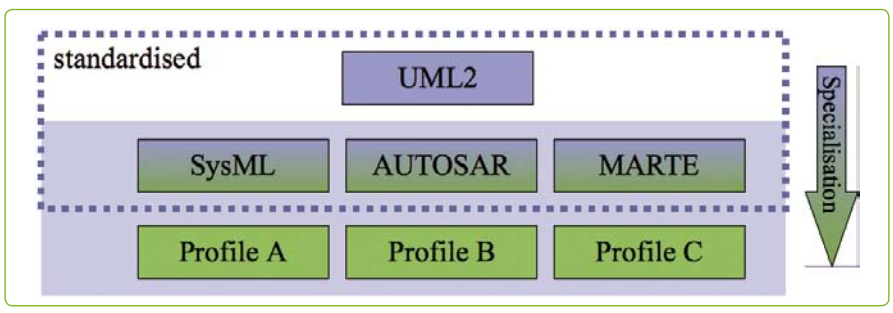


Abb 1: UML-basierte DLSs

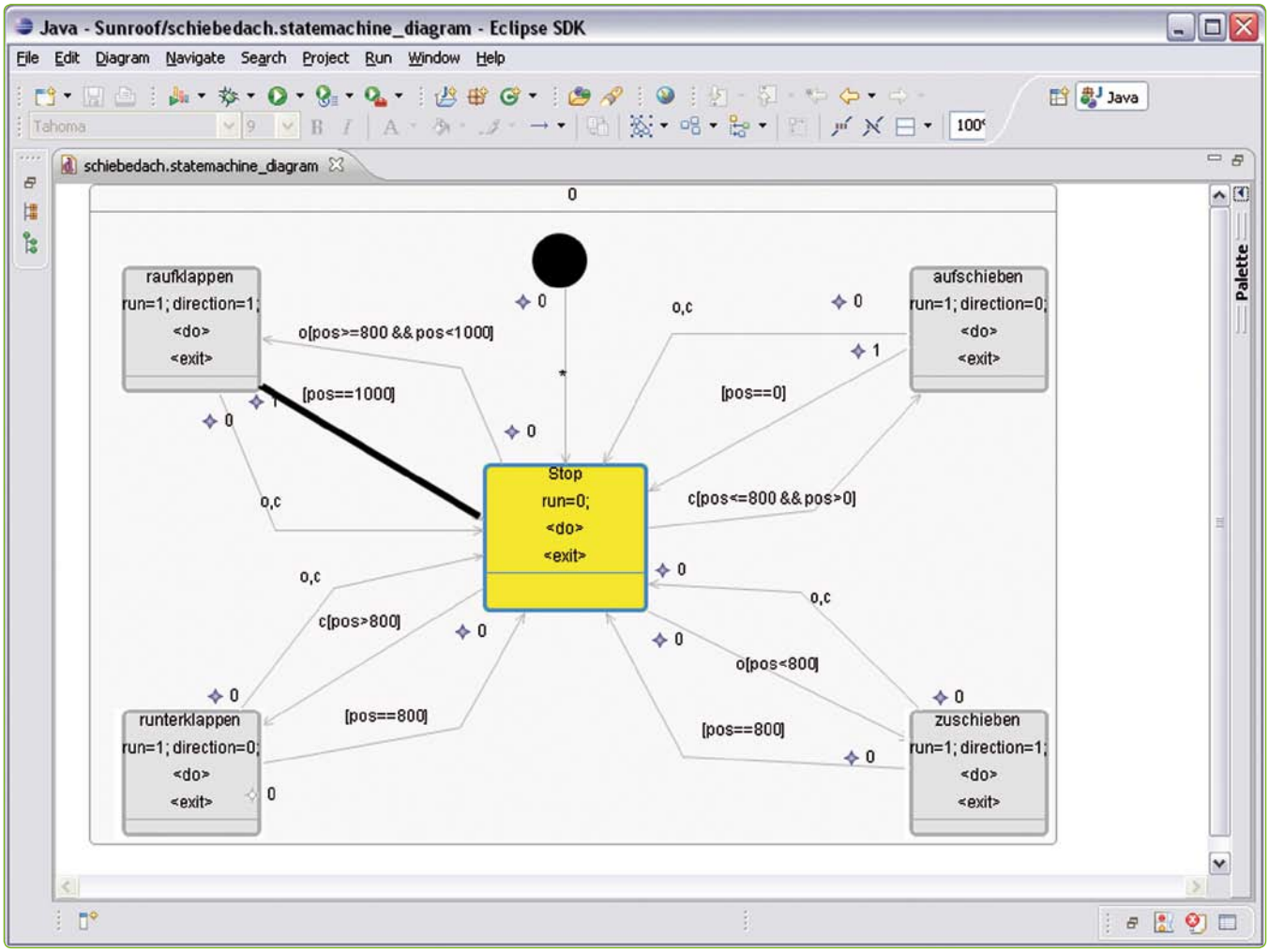


Abb. 2: Simulation im Statemachine-Editor

Performanz der Validierung. Nimmt ein Entwickler Änderungen am Modell vor, will er zeitnah wissen, wie sich die Änderungen auf das übrige Modell auswirken. Viel Zeit kann man gewinnen, indem die Regelsätze an die jeweilige Situation angepasst werden. Häufig ist es gar nicht notwendig, einen kompletten Regelsatz zu überprüfen. Zum einen sollten natürlich nur solche Regeln überprüft werden, die mit der Änderung in Zusammenhang stehen; zum anderen will man vielleicht nur spezifische Regeln ausführen, die nur in der Sicht, die der Entwickler auf das Modell hat, Auswirkungen zeigen. Auch bei der Entwicklung von Produktlinien, die im Bereich eingebetteter Systeme weit verbreitet sind, sind für die unterschiedlichen, konkreten Produkte unter Umständen unterschiedliche Regelsätze notwendig.

Ein Validierungs-Framework sollte diesen Anforderungen genügen. Im Eclipse-Umfeld bietet beispielsweise EMF eine Komponente zur Validierung. Der Validierungsalgorithmus ist bereits vorgegeben. Die notwendigen Regeln können relativ einfach in Java implementiert und in das Framework integriert werden, sodass in kürzester Zeit eine Validierung des Modells zur Verfügung steht. Einzelne oder ganze Gruppen von Regeln können durch den Benutzer an- und abgeschaltet werden, was der Performanz zu Gute kommt. Die Schnittstelle zur Definition von Regeln ist offen. Neben der Implementierung der Regeln in Java können hier Regeln auch durch die Object Constraint Language (OCL), die ein Bestandteil der UML ist, ausgedrückt werden. Möglich ist aber auch eine eigene, domänenspezifische Sprache, mit deren Hilfe die Benutzer des Modells eigenständig und ohne Java-Programmierkenntnisse Regeln definieren können. Eine weitere Alternative im Eclipse-Projekt ist Check aus dem openArchitectureWare-Projekt [4]. Check ermöglicht die Validierung beliebiger Modelle und ist nicht – wie die EMF Validierungskomponente – auf MOF-kompatible (genauer gesagt ECORE-kompatible) Metamodelle festgelegt.

Die bereits erwähnte Integration von Metamodellen unterschiedlicher, vorhandener Werkzeuge, bietet gerade bei der Validierung der Modelle große Vorteile. Beziehungen zwischen verschiedenen Modellen wie Blockschaltdiagrammen und Statecharts können durch die Integration automatisch auf ihre Konsistenz hin überprüft werden. Die mühselige und vor allem fehleranfällige manuelle Überprüfung der Konsistenz der verschiedenen Modelle oder die Sicherstellung der Konsistenz durch Konventionen ist somit überflüssig. Die vorhandenen Werkzeuge ließen sich also sehr viel effizienter nutzen. Gerade für eingebettete Systeme ist der Beweis sicherheitskritischer Eigenschaften oft zur Zulassung der Systeme notwendig. Ein formaler Beweis solcher Eigenschaften ist auf Quelltextebene von C oder Assembler nur schwer mög-

lich. Dagegen wird in der Praxis immer häufiger eine formale Verifikation von Modellen mittels Modelchecking verwendet. Werkzeuge wie Spin [11] oder Uppaal [12] stammen aus dem universitären Umfeld, haben aber auch schon in der Praxis Anwendung gefunden. Spin wurde beispielsweise erfolgreich in verschiedenen Weltraummissionen wie Deep Space 1, Cassini oder dem Mars Exploration Rover eingesetzt. Es erlaubt auch die Verwendung von C-Code in den zu verifizierenden Modellen. Das neuere Werkzeug Uppaal unterstützt dagegen die Modellierung, Validierung und Verifikation von Echtzeitsystemen auf Basis von Timed Automata. Die Verifikation hat aber auch Haken: Auch wenn das Modell verifiziert wurde, muss noch immer sichergestellt werden, dass die Codegenerierung korrekt ist. Außerdem ist die Größe der Modelle, die heutzutage durch Werkzeuge automatisch verifiziert werden können, noch sehr begrenzt.

Verhaltensmodellierung, Simulation und Visualisierung

Bei der Entwicklung eingebetteter Systeme ist die Verhaltensmodellierung von besonderer Bedeutung. Zum Einsatz kommen dazu beispielsweise Blockschaltdiagramme, Zustandsdiagramme oder Message Sequence Charts [13]. Im Unterschied zur Entwicklung von Anwendungssystemen müssen für eingebettete Systeme teilweise Echtzeitanforderungen berücksichtigt werden. Zur Überprüfung der Korrektheit lassen sich wie gezeigt Validierung und begrenzt formale Verifikation einsetzen. Ein ergänzendes Mittel zur Überprüfung und zur Reflektion stellt die Simulation dar. Bei der Simulation wird das durch das Modell beschriebene Verhalten anhand

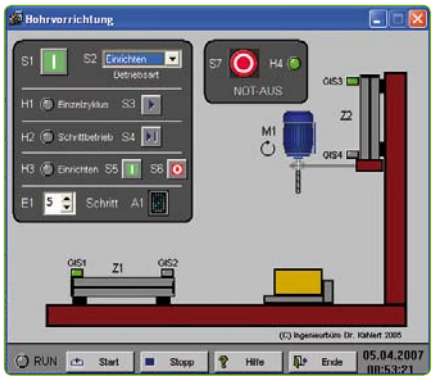


Abb. 3: Simulation mit angepasster Visualisierung

von Szenarien ausgeführt. Eine Simulation ist umso wichtiger, je aufwändiger ein Test auf der Zielplattform in der späteren Zielumgebung ist. Im Extremfall kann dies sogar unmöglich sein, denkt man beispielsweise an die Steuerung eines Satelliten. Bei der Simulation unterscheidet man zwischen der Simulations-Engine und dem Simulations-Frontend. Während die Engine Szenario und Modell erhält und die Einzelschritte berechnet, dient das Frontend zur Visualisierung und Interaktion mit dem Benutzer, um z.B. Szenariowerte oder Simulationsparameter zu ändern oder Zwischenzustände zu inspizieren. Die Visualisierung kann entweder in der Modellsicht oder in einer speziell an die Zielumgebung angepassten Sicht erfolgen. In beiden Fällen ist eine Kommunikation zwischen Engine und Frontend über eine definierte Schnittstelle notwendig. Besonders herausfordernd wird dies, wenn sich die Simulation über unterschiedliche Diagrammtypen erstreckt. Im Forschungsprojekt MDA4E [14] haben wir dies für die Verbindung von Blockschaltdiagrammen und Zustandsdiagrammen umgesetzt (Abb. 2). Dabei wurde deutlich, dass heutige Editoren für UML-Zustandsdiagramme keine definierte Schnittstelle für die Simulation zur Verfügung stellen. So fiel die Entscheidung, zunächst mithilfe von GMF einen eigenen Editor bereitzustellen, um Erfahrungen mit der benötigten Simulationsschnittstelle zu gewinnen. Im Ergebnis lassen sich nun Steuerungen simulieren, die in Blockschaltdiagrammen definiert werden. Einzelne Blöcke können durch Zustandsdiagramme verfeinert werden. Die Simulation kann entweder in den Modelleditoren oder in einer speziell angepassten Visualisierung sichtbar gemacht werden.

> Die Verhaltensmodellierung ist bei eingebetteten Systemen von großer Bedeutung

Festzuhalten ist also, dass Eclipse bereits heute die notwendigen Basiswerkzeuge bereitstellt, um Simulations-Frontends zu erstellen und an Simulations-Engines anzubinden. Allerdings fehlt bislang eine Eclipse-Strategie, wie Simulationsschnittstellen der Editoren grundsätzlich aussehen sollten. Dies behindert die Nutzung vorhandener Editoren, die sich für standardisierte Diagrammarten mittlerweile reichlich finden lassen.

Codegenerierung

Die Codegenerierung im Bereich der eingebetteten Systeme unterscheidet sich im eigentlichen Sinne nur wenig von der klassischen Codegenerierung in der Applikationsentwicklung. Aus Modellen, die auf Komponentenstandards wie zum Beispiel Autosar [15] oder MSR [16] aus der Automobilindustrie oder SMP2 [17] und SMP [18] im Bereich der Simulation aufbauen, werden die benötigten Artefakte generiert. Dies sind üblicherweise C- oder C++-Quellcode-, Header- oder auch Konfigurationsdateien. Es sind jedoch einige besondere Anforderungen zu beachten.

Wie bereits im vorigen Abschnitt angedeutet, reicht es nicht aus, wenn das Modell zwar korrekt, die Codegenerierung jedoch fehlerhaft ist. Ein wichtiges Thema bei sicherheitskritischen Systemen ist deshalb die Zertifizierung. Zertifiziert werden kann entweder der konkrete, generierte Quellcode oder aber die Codegenerierung selbst. Bei der Zertifizierung des generierten Quellcodes kommen z.B. Verfahren zur Messung der Codeüberdeckung durch Tests zum Einsatz. Die Zertifizierung der Codegenerierung ist zwar komplexer, muss dagegen aber nur einmal erfolgen. Zurzeit sind Verfahren zur Verifikation von Codegeneratoren Gegenstand der Forschung.

Durch die extrem große Vielfalt der Zielplattformen ist auch die Produktlinienentwicklung ein wichtiges Thema. Der Quellcode für die konkreten Produkte unterscheidet sich dabei in der verwendeten Zielsprache oder Hardware der Plattform, den verwendeten Bibliotheken oder auch in den Teilen des Modells, aus denen der Code generiert wird. Die Codegenerierung sollte dies unterstützen, ohne spezielle Codegeneratoren für jede der Zielplattformen entwickeln zu müssen. So ist vor allem ein modularer Aufbau der Templates, die zur Codegenerierung verwendet werden, hilfreich. Die für die Zielplattform

benötigten Templates können dadurch einfach ausgewählt und zusammengesetzt werden. Auch modifizierbare Templates verringern den Aufwand. Die Modifizierbarkeit kann beispielsweise durch aspektorientierte Programmierung (AOP) erreicht werden, bei der zusätzliche Informationen je nach Zielplattform in die Templates eingebracht werden. Gute Ansätze zur Unterstützung von Produktlinienaspekten bietet Xpand aus dem openArchitectureWare-Projekt. Ausführlicher wird dieses Thema im Beitrag von Markus Völter und Iris Grohe in [19] behandelt.

Die in eingebetteten Systemen verwendete Hardware ist wegen möglichst niedriger Kosten oder niedrigem Energieverbrauch meist sehr eingeschränkt in ihrer Leistungsfähigkeit. Es mangelt an Speicherplatz und hoher Rechengeschwindigkeit. Die Effizienz des generierten Codes ist deshalb sehr wichtig. Die gängige Vorstellung, nur manuell programmierter Code sei effizient, ist mittlerweile so nicht mehr haltbar. Generierter Code ist dem manuell programmierten Code zum Teil sogar überlegen. Codemuster für wiederkehrende Konstrukte oder spezifische Anpassungen der Codegenerierung an die Zielplattform helfen beispielsweise bei der Optimierung. Um all die bei der Codegenerierung eingesetzten Optimierungen dagegen auch bei der manuellen Programmierung verwenden zu können, müssten die Entwickler umfangreich geschult werden. Aber auch dann ist nicht sichergestellt, dass jeder Entwickler alle bekannten Optimierungen einsetzt. Werkzeuge zur Codegenerierung sind weit verbreitet. Im Eclipse-Umfeld werden unter anderem JET [20] und openArchitectureWare Xpand [OAW] eingesetzt. Diese Codegeneratoren sind relativ universell einsetzbar.

Fazit

Viele Herausforderungen bei der modellgetriebenen Entwicklung für eingebettete Systeme sind uns aus der modellgetriebenen Entwicklung von Anwendungssystemen bekannt. Daher können dort etablierte Methoden und Werkzeuge auch für diese Domäne bereits jetzt erfolgreich genutzt werden. Insbesondere Eclipse und die zahlreichen kommerziellen und nicht kommerziellen Angebote in den Bereichen Editieren, Codegenerierung, Verifikation, Validierung und auch Simulation ermöglichen einen schnellen Start mit einer integrierten Werkzeugumgebung. Dennoch gibt es noch Bereiche, in denen eigene Integrations- und Anpassungsarbeit notwendig ist. Hier sind insbesondere die integrierte Verwendung unterschiedlicher Metamodelle, DSLs und Editoren sowie die Einbindung von Simulations- und Verifikations-Engines zu nennen.

Wolfgang Neuhaus ist im Vorstand der itemis AG und beschäftigt sich seit vielen Jahren mit modellgetriebenen Entwicklungsmethoden, zunehmend auch für eingebettete Systeme. Kontakt: wolfgang.neuhaus@itemis.de.

Benedikt Niehues ist Berater und Softwarearchitekt bei itemis products & solutions GmbH & Co KG. Er hat sich auf modellgetriebene, generative Softwareentwicklung spezialisiert. Kontakt: benedikt.niehues@itemis.de.

Lothar Wendehals ist Berater und Softwarearchitekt bei der itemis AG. Seine Spezialgebiete sind seit vielen Jahren modellbasierte Entwicklungsumgebungen und Reverse Engineering. Kontakt: lothar.wendehals@itemis.de.

>> Links & Literatur

[1] www.mathworks.com
[2] modeling.telelogic.com/products/rhapsody/index.cfm
[3] www.etas.com/de/products/ascet_software_products.php
[4] www.openarchitectureware.org
[5] www.eclipse.org/m2m/atl/
[6] www.sysml.org
[7] www.eclipse.org/modeling/
[8] www.eclipse.org/modeling/emf/
[9] www.eclipse.org/modeling/gmf/
[10] www.topcased.org
[11] spinroot.com/spin/whatispin.html
[12] www.uppaal.com
[13] www.sdl-forum.org/MSC/
[14] opensource.itemis.de/mda4e/
[15] www.autosar.org
[16] www.msr-wg.de
[17] portal.vega.de/smp/
[18] www.ecss.nl, WG ECSS E40-07
[19] Florian Fieber, Wolfgang Neuhaus, Roland Petrasch (Hrsg): Modellbasierte Software-Entwicklung für eingebettete Systeme, Logos Verlag Berlin 2007
[20] www.eclipse.org/modeling/m2t/?project=jet#jet