Mit DVD

iX DEVELOPER

4 192045 312906 02

€ 12,90

Österreich € 14,20
Schweiz CHF 25,80
Benelux € 14,80
Italien € 16,80

2/2014

Sonderdruck

Auf der Heft-DVD:

Massig Software für Entwickler

IDEs: Code::Blocks, CodeLite, Eclipse, mbeddr, Visual Studio

Testversionen: Cantata, Intel System Studio

Tools: AVR Libc, GCC, GNAT GPL, GNU Binutils, MAST, openHAB, ProjectLibre, Valgrind

Bücher: Testen von Software und Embedded Systems (komplett), Software-Test für Embedded Systems (Auszüge)

Sponsored Software: Enterprise Architect 10, IDE für Raspberry Pi

Embedded Software

Ada, C, C++, MISRA C, Java, Pascal

Wie Programmiersprachen Echtzeit, Safety und Kompaktheit meistern

Modellierung

Systementwicklung mit SysML und UML

Embedded-Markt im Umbruch

Arduino, Raspberry Pi und BeagleBone

Industrie 4.0 und das Internet der Dinge

Durchbruch für die Heimautomatisierung

Qualitätssicherung

Wichtige Tools, Standards und Methoden

Debugging von Multicore-Systemen

Datenträger enthält **Info- und Lehrprogramme** gemäß § 14 JuSchG

präsentiert von: heise Developer www.heise-developer.de

System- und Softwareentwicklung mit SysML und UML für eingebettete Systeme

Ein Bild sagt mehr als tausend Worte

Oliver Alt

Seit 2007 steht mit der SysML eine standardisierte Modellierungssprache zur Verfügung, die es erlaubt, Hardware- und Softwareanteile eingebetteter Systeme grafisch zu spezifizieren. Darüber hinaus lassen sich solche SysML- durch UML-Softwaremodelle erweitern. Damit kann man die Vorteile beider Sprachen kombinieren und eine modellbasierte Entwicklung über den gesamten Produktlebenszyklus realisieren – angefangen bei der Anforderungsentwicklung bis hin zur Generierung ausführbaren Codes.

Die Grundprinzipien einer technischen Entwicklung sind immer die gleichen: Es geht darum, ein aus einzelnen Komponenten bestehendes System so zu entwerfen und zu realisieren, dass durch das Zusammenspiel der einzelnen Komponenten eine gewünschte Funktionalität erfüllt wird. In der Vergangenheit wurde das mechanisch, hydraulisch oder pneumatisch gelöst. Mit dem Aufkommen der Elektrotechnik und Elektronik kam eine neue Dimension hinzu. Nun ließen sich Funktionsabläufe nicht mehr nur rein mechanisch steuern oder regeln, sondern auch elektronisch. Programmierbare Microcontroller erweiterten in den letzten Jahrzehnten die Möglichkeiten um die der Softwareentwicklung.

Software bietet den großen Vorteil, leicht und günstig änder- und erweiterbar zu sein. Daher werden heute fast nur noch softwaregesteuerte Geräte entwickelt, da die Flexibilität im Gegensatz zu rein elektronischen Steuerungen um ein Vielfaches höher ist. Sobald man nun Elektronik und Mechanik kombiniert, spricht man von mechatronischen Systemen. Sind

diese zusätzlich mit einem Mikroprozessor mit Software, zum Beispiel in Form eines Microcontrollers, ausgestattet, hat man es mit einem eingebetteten System zu tun. Die Entwicklung eines solchen eingebetteten, mechatronischen Systems erfordert daher Wissen im Bereich der Mechanik, der Elektronik und der Softwaretechnik.

Die Fortschritte in der Fertigungstechnik erlauben es, in eingebetteten Systemen zunehmend leistungsstärkere Mikrocontroller mit mehr Speicher zu immer geringeren Kosten einzusetzen. Das geht einher mit einem wachsenden Funktionsumfang der Systeme, umgesetzt vorwiegend durch größere Softwareanteile. Damit steigt natürlich auch der Entwicklungsaufwand und die Komplexität der Systeme nimmt stark zu. Das führt dazu, dass die Gesamtfunktionalität mit konventionellen Entwicklungsmethoden – bei denen zumeist textuelle Dokumente und einzelne Zeichnungen Komponenten beschreiben – nur noch schwer oder unzureichend zu überblicken ist. Allein die Spezifikationstexte eines eingebetteten Systems können heute mehrere Hundert bis mehrere Tausend Seiten Text umfassen.

Das Problem steigender Komplexität bei eingebetteten Systemen ist seit Jahren ein Thema und braucht innovative Ideen. Die Softwareentwicklung adressiert dieses Ziel bereits seit einiger Zeit unter der Disziplin des Software Engineering, der Entwicklung und Anwendung ingenieurwissenschaftlicher Methoden im Bereich der Softwareentwicklung. Dabei erwies sich die modellbasierte Entwicklung, die Nutzung von Modellen zur Beschreibung von Zusammenhängen, anstelle der bislang üblichen Textdokumente als hilfreich und zukunftsweisend.

Die Arbeit mit Modellen ist grundsätzlich nicht neu, sondern wird seit Jahrtausenden zum Beispiel in der Architektur als Hilfsmittel eingesetzt. Modelle im Softwarebereich haben allerdings einen großen Vorteil etwa im Vergleich mit einem Architekturmodell: Da Softwaremodelle zumeist in einer für Rechner verarbeitbaren Form vorliegen, lassen sich diese direkt dazu nutzen, ausführbare Software zu generieren. Manuelle Codierung wird damit ein Stück weit oder sogar komplett ersetzt. Modelle im Bauingenieurwesen können das nicht: Ein Haus wird, basierend auf einem Modell, weiterhin durch Baufirmen manuell errichtet.

Zur weiteren Verbesserung der Systementwicklung entstanden neue Disziplinen wie das Anforderungs-Engineering, das mit einem strukturierten Vorgehen die Anforderungen an eine Software oder ein System dokumentiert und nachvollziehbar den Projektbeteiligten zugänglich macht. Damit bleiben die Gründe für eine Entwicklung auch nach Fertigstellung eindeutig nachvollziehbar.

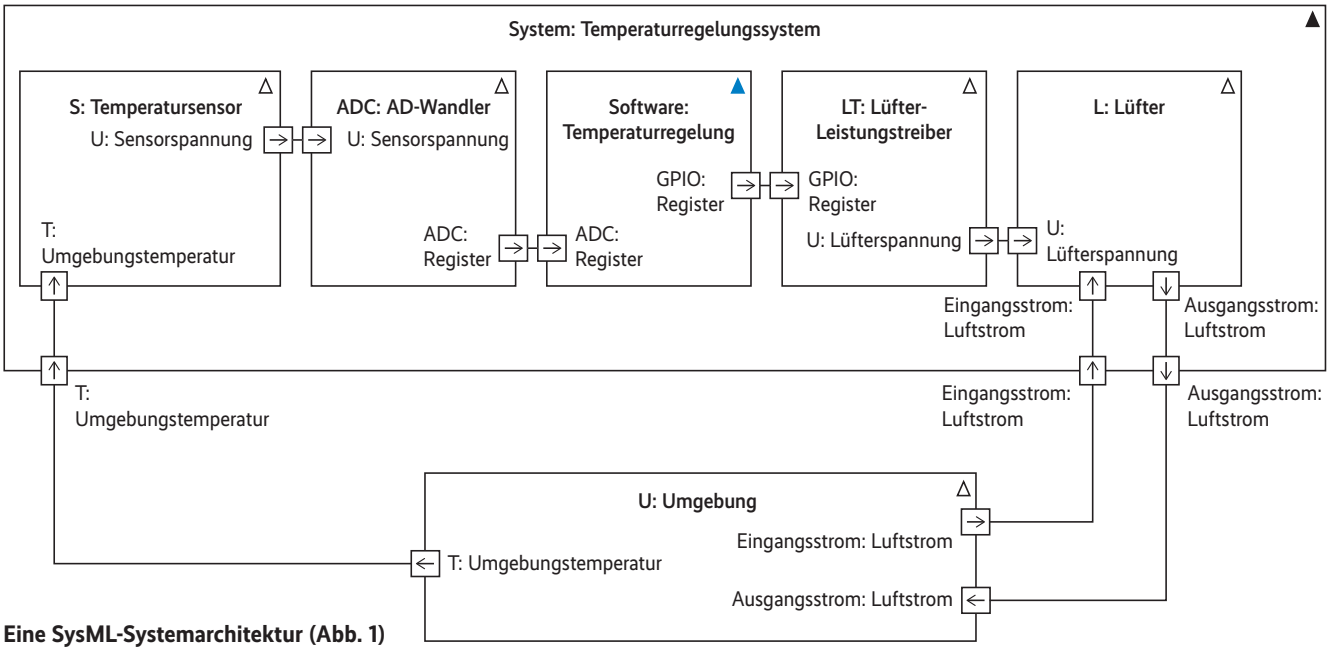
Modellbasierte Entwicklung in mechatronischen Systemen

Bedingt durch den Erfolg der modellbasierten Entwicklung in der Softwaretechnik liegt es nahe, den Ansatz auf die Entwicklung mechatronischer Systeme zu übertragen. Dabei nutzt man auch Modelle im Systems Engineering, um die Zusammenhänge innerhalb eines Hardware-/Softwaresystems zu beschreiben. Aber wie sehen nun solche Modelle aus?

Im Bereich der Softwaretechnik wurde bereits in den 1990er-Jahren die Unified Modeling Language (UML) als Modellierungsstandard für vorwiegend objektorientierte Software definiert. Sie ist eine grafische Modellierungssprache, besteht also aus grafischen Symbolen und Verbindungen zwischen diesen. Die Symbole haben bestimmte Bedeutungen, um so die Zusammenhänge zu beschreiben.

Aufbauend auf der UML ist etwa 2007 die Systems Modeling Language (SysML) [1] entstanden. Sie übernimmt Teile der UML, lässt aber auch Dinge weg und ergänzt zusätzlich den Sprachumfang um neue Elemente, die aus der Systemtechnik kommen und für die Systementwicklung relevant sind. Insbesondere bietet die SysML erstmals die Möglichkeit, textuelle Anforderungen standardisiert in ein Modell einzubringen. Diese lassen sich mit anderen Elementen wie Systemkomponenten, Schnittstellen oder auch Testfällen verlinken und gewährleisten so die Nachverfolgbarkeit (Traceability) zwischen den Elementen.

Um die Konzepte der modellbasierten Systementwicklung zu veranschaulichen, soll als Beispiel eine Temperaturüberwachung und -regelung dienen, die aufgrund der gemessenen Um-



Eine SysML-Systemarchitektur (Abb. 1)

gebungstemperatur einen Lüfter zwecks Kühlung aktiviert. Der gemessene Schwellwert zur Aktivierung soll 50° C betragen. Ausgehend von diesen ersten Anforderungen beginnt die Systementwicklung nun, sowohl weitere Anforderungen als auch eine Systemarchitektur zur Implementierung zu entwickeln. Die Architektur lässt sich mit einem SysML-Diagramm darstellen. Ein solches Architekturdiagramm für das Beispielsystem stellt Abbildung 1 dar. Das hier verwendete Entwurfskonzept heißt Wirkkette. Das bedeutet, dass man Hardware- und Softwarekomponenten gemeinsam als zusammenwirkende Einheiten darstellt. Software ist im Beispiel durch ein blaues Dreieck markiert, Hardware durch ein weißes. Zusätzlich gibt es als umgebende, rein logische Komponente noch eine schwarz markierte, den Kontext abgrenzende Komponente. Logische Komponenten zusätzlich in die Architektur zu bringen, hat den Vorteil, dass man beispielsweise Anforderungen auch mit solchen Komponenten verlinken kann.

Traceability ist wichtig

Abbildung 2 zeigt ein SysML-Diagramm, das Systemkomponenten, einen Testfall und textuelle Anforderungen enthält. Über die Beziehungen (*satisfy*, *deriveReqt* und *verify*) wird die Nachverfolgbarkeit (Traceability) dokumentiert.

Die Entwicklung eines solchen Diagramms vollzieht sich dabei typischerweise in mehreren Schritten. Man beginnt mit einer oder mehreren Anforderungen, entwickelt eine Architektur als Lösung, verlinkt die Anforderungen und sucht schließlich nach zusätzlichen Anforderungen aufgrund der getroffenen Architekturentscheidungen. Mit dem weiteren Fortschreiten der Entwicklung erfolgt eine Verfeinerung und Detaillierung der Modelle. Hier lassen sich für die Softwaremodellierung Konzepte der UML hinzunehmen, die die SysML-Modelle im Bereich der implementierungsnahen Softwaremodellierung geeignet ergänzen.

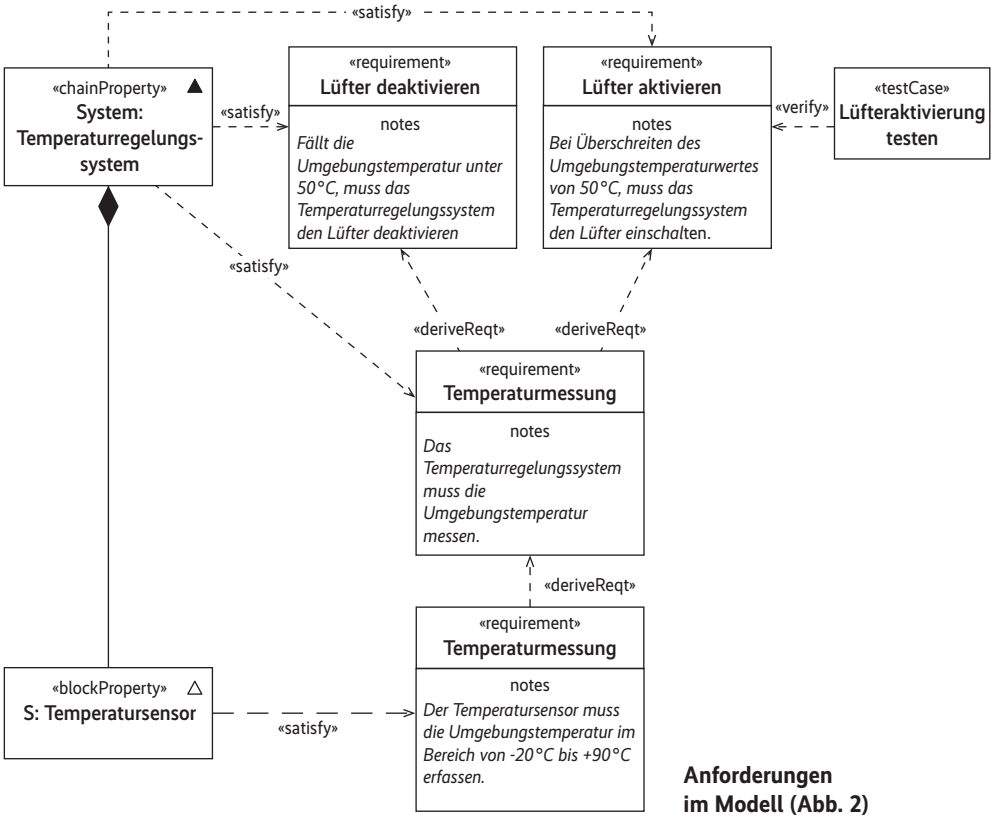
Im Bereich der eingebetteten Systeme ist die nicht objektorientierte Programmiersprache C zumeist die erste Wahl, gerade im Hinblick auf die beschränkten Ressourcen eines solchen Systems. Arbeitsspeicher von wenigen Kilobyte bis einigen Byte sind hier keine Seltenheit. Objektorientierte Sprachen wie Java oder C# sind in diesem Fall nicht geeignet, da ein Grundkonzept der Objektorientierung auf dynamische Speicherzuweisung zur Laufzeit setzt. Sobald eine Objektinstanz erzeugt wird, ist für diese entsprechender Arbeitsspeicher zu reservieren und am Ende der Objektlebensdauer wieder freizugeben. Diese Sprachen sind also auf einen entsprechend großen Arbeitsspeicher und dynamische Speicherverwaltung angewiesen.

UML und die Randbedingungen

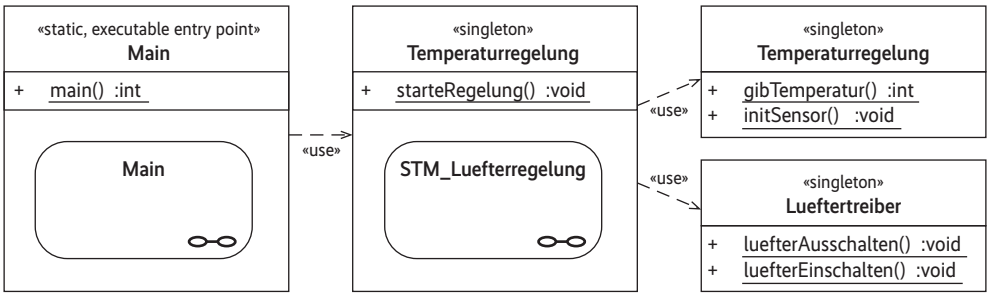
Bei der Entwicklung objektorientierter Software, zum Beispiel im Bereich der PC-Anwendungen, hat sich inzwischen neben handgeschriebenem Code auch die Codegenerierung aus Modellen in Teilen etabliert. Zum Erstellen solcher Modelle kommen meist grafische Modellierungssprachen wie UML zum Einsatz, die die Struktur und das Verhalten der Software formal spezifizieren. Codegeneratoren ermöglichen es schließlich, aus diesen Modellen Code abzuleiten.

Im Umfeld der Entwicklung eingebetteter Systeme findet die UML bislang eher selten Verwendung, da diese Systeme eben nicht objekt-, sondern funktionsorientiert entwickelt und programmiert werden. Und trotzdem ist es durchaus möglich, UML und Codegenerierung in C auch für die Entwicklung eingebetteter Systeme zu nutzen. Dafür bedarf es aber einiger Randbedingungen:

1. Nicht alle UML-Möglichkeiten lassen sich für C-Codegenerierung nutzen: Spezielle Konstrukte der Objektorientierung wie Vererbung, Polymorphismus und dynamische Objektinstanziierung haben keine direkte Entsprechung in C. Daher finden diese bei der Modellierung auch zumeist keine Verwendung.
2. Ein Mapping der UML-Konstrukte auf die funktionsorientierte Sicht von C ist notwendig: Das bedeutet zum Beispiel, dass das UML-Element der Klasse auf eine C-Datei abgebildet wird. Weiterhin lassen sich Schnittstellen, die bei C typischerweise in einer Header-Datei (*.h*-Datei) definiert werden, entweder implizit durch die Klasse mitdefinieren oder aus explizit modellierten UML-Interface-Elementen generieren. Somit entstehen aus einer Klasse je nach Modell manchmal zwei Dateien (*.c*- und *.h*-Datei) oder nur eine (*.c*-Datei).
3. Die Modellierung sollte nicht den vollen Umfang der Pro-



Anforderungen im Modell (Abb. 2)



Statische Softwarestruktur des Beispielsystems (Abb. 3)

grammiersprache C unterstützen: Was zunächst wie ein Nachteil klingt, wird sich auf lange Sicht als Vorteil erweisen. C ist vom Konzept her vielfältig hinsichtlich der Möglichkeiten der Sprache. Insbesondere der explizite Umgang mit Zeigern (Pointer) auf Variablen und Funktionen sowie die damit erlaubte Pointer-Arithmetik gewähren es dem Programmierer, tief in die Abbildung von Daten im Speicher einzugreifen. Diese Konstrukte unterstützen objektorientierte Sprachen wie Java und C# aufgrund ihrer Fehleranfälligkeit bewusst nicht mehr. Trotzdem kann man mit diesen Sprachen ebenfalls alles erreichen. Daher gibt es im C-Umfeld auch andere Wege zum Ziel. Eine explizite Modellierung von Pointern ist für die Modellierung also nicht vorzusehen.

4. Bestimmte C-Konstrukte erfordern UML-Erweiterungen: Nicht alle zumindest benötigten Dinge aus C lassen sich durch die Standard-UML direkt abbilden. Glücklicherweise definiert die UML hier einen Mechanismus, um UML-Elemente mit zusätzlichen neuen Eigenschaften auszustatten: den UML-Profilmechanismus. Erweiterte UML-Elemente werden durch in französischen Anführungszeichen («») stehende Schlüsselwörter (Stereotypen) gekennzeichnet. Neue Eigenschaften lassen sich mit sogenannten Tagged Values definieren.

Statische Struktur

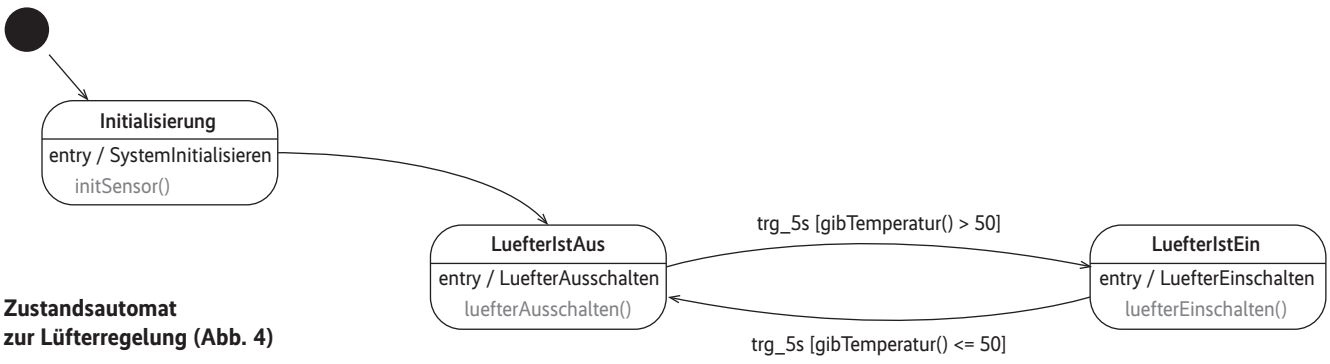
Die Modellierung der Software beginnt typischerweise mit der Definition der statischen Struktur. In der UML benutzt man dafür Klassendiagramme und modelliert so Klassen und Schnittstellen des Systems. Diese Elemente beschreiben den Namen und darunter in den Bereichen die Attribute, Operationen und das Verhalten der Klasse. Für die C-Codegenerierung erfolgt die Abbildung der Attribute auf Variablen einer C-Datei und der Operationen auf Funktionen.

Schnittstellenelemente (Interface) bildet man auf Variablendefinitionen und Funktionsprototypen in eine Header-Datei, die durch die C-Datei inkludiert (*#include*) wird, ab. Ob eine UML-Schnittstelle in einer C-Datei genutzt oder implementiert wird, legen die UML-Beziehungen „Usage“ oder „Realize“ fest. Fehlt eine explizite Schnittstellendefinition, entsteht eine Header-Datei automatisch aus der Klasse und deren Attribut- und Operationsdefinitionen.

Attribute sind in der UML immer an eine bestimmte Klasse gebunden. Um sie auch im generierten C-Code an ein bestimmtes Modul zu binden, lässt sich die UML-Klasse als C-Struktur abbilden. Die Attribute der Klasse werden so als Variablen der Struktur definiert und nicht frei im C-Modul. Die Verwendung der Struktur als Kapselung ermöglicht es, ein Modul und dessen Funktion auch mehrfach als Variable zu definieren und zu verwenden, ähnlich wie man es in der Objektorientierung mit mehreren Instanzen derselben Klasse tut. Der Unterschied ist nun aber, dass der Speicher für die Struktur schon zur Programmierzeit reserviert wird. Es gäbe in C zwar auch die Funktion *malloc()* (Memory Allocate), diese ist jedoch in kleinen eingebetteten Systemen aufgrund des begrenzten Speichers nicht verwendbar.

Wenn das Konzept der mehrfachen Nutzung eines C-Moduls so nicht gewünscht ist, lässt sich das durch Kennzeichnung der Klasse zum Beispiel mit dem Stereotyp *«singleton»* – in Anlehnung an ein ähnliches objektorientiertes Entwurfsmuster – verhindern.

Abbildung 3 zeigt die statische Struktur der Softwaremodellierung des Beispielsystems. Die eigentliche Funktion steckt dabei in einem Zustandsautomaten, der der Klasse *Temperaturregelung* zugeordnet ist. UML bietet die Möglichkeit, statische Konstrukte wie Klassen und Aspekte der dynamischen Verhaltensbeschreibung wie Zustandsautomaten und Aktivitäten direkt zu verknüpfen. Damit kann man mit einem solchen Modell eine Software vollständig in Struktur und Verhalten beschreiben und einen vollständigen Code daraus generieren.



Zustandsautomat zur Lüfterregelung (Abb. 4)

Welt voller Diagramme

Durch die Modellierung der Struktur kann ein Codegenerator bereits Header-Dateien und leere Funktionsrümpfe generieren. Interessanter wird es aber, wenn im Modell auch das Verhalten der Software beschrieben ist. Am interessantesten sind hierbei das Zustands- und das Aktivitätsdiagramm. Die Zustandsdiagramme in UML gehen zurück auf die Ende der 1980er-Jahre von David Harel definierten Statecharts. Sie unterstützen hierarchische Zustandsautomaten mit entsprechenden Übergängen und der Ausführung von Aktivitäten beim Betreten, Durchführen oder Verlassen eines Zustandes sowie während des Zustandsübergangs. Hier zeigt sich die Verbindung mit den Aktivitätsdiagrammen der UML. Sie leiten sich von den Flussdiagrammen ab, die seit den 1960er-Jahren verwendet werden. Aktivitätsdiagramme beschreiben durchzuführende Aktionen, Verzweigungen, Schleifen et cetera.

Zusätzlich zur direkten Verknüpfung statischer und dynamischer Modellelemente ermöglicht es die UML, „im Notfall“ auch direkt Code bei einer Operation zu hinterlegen, der eins zu eins in den generierten Code übernommen wird. So lassen sich auch Codeteile und Konstrukte nutzen, die der Codegenerator so (noch) nicht auf Modellebene unterstützt. Nicht zuletzt lässt sich sogar Legacy-Code aus Altsystemen einbeziehen.

Das funktionale Softwareverhalten der Temperaturregelung ist durch den in Abbildung 4 dargestellten Zustandsautomaten modelliert. Es wird alle fünf Sekunden überprüft, ob die Temperatur den Wert 50 über- oder unterschreitet. Wenn ja, wird der Zustand gewechselt und beim Eintritt der Lüfter aktiviert oder deaktiviert.

Codegenerierung und Debugging

Bei der Codegenerierung entstehen aus einem Zustandsautomaten Datenstrukturen und C-Module, die per Switch/Case-Anweisung die Abarbeitung der Zustände als Codemodul umsetzen. Aktivitätsdiagramme wiederum werden zu Funktionsaufrufen sowie Verzweigungen (if, if-else, else) und Schleifen. Damit unterstützt die UML weite Teile des Sprachstandards heutiger Programmiersprachen und erlaubt es, nahezu alle Herausforderungen an eine Software für eingebettete Systeme zu meistern.

Ein noch zu behebender Mangel der Code-Generierung aus Modellen ist die manchmal umständliche Art, das gewünschte Verhalten im Modell abzubilden. Das ist zwar möglich, jedoch manchmal mit erheblichem Modellierungsaufwand verbunden. Ein ähnliches Problem bestand vor ein paar Jahren auch noch bei der klassischen Programmierung, als es noch keine integrierten Entwicklungsumgebungen gab. Die nun dort vorhandenen Hilfsmittel für Programmierer wie IntelliSense in Visual Studio und umfangreiche Suchfunktionen machen es inzwischen leicht, sich auch in einem großen Softwareprojekt zurechtzufinden. Diese Anwenderunterstützung ist für die Modellierung heute noch nicht in vollem Umfang vorhanden. Die Arbeit an ähnlichen Konstrukten auch auf der Modellierungsebene ist aber in vollem Gange und wird die Erstellung und Wartung von Programmcode möglich.

Ein weiterer Schritt in eine umfassende Nutzung des Modells für die Softwareentwicklung ist das Debugging der Software auf der Modellebene. Wenn man schon Code aus einem Modell generiert, warum sollte man nicht auch die Fehlersuche auf der Modellebene ausführen statt im generierten Code?

Dazu werden die Modelle mit dem generierten Code und entsprechenden Debug-Schnittstellen verknüpft. So lässt sich der Ablauf des Codes im Zielsystem in Form einer Visualisierung direkt im Modell verfolgen. Breakpoints lassen sich auch auf Modellebene setzen, um Zustände von Variablen direkt im Modell anzuzeigen beziehungsweise diese zu modifizieren. Auch eine schrittweise Ausführung von Aktionen oder Zustandsübergängen im Modell ist vorstellbar.

Hat man den Fehler lokalisiert, wird dieser direkt im Modell behoben und dann der Code neu generiert. Das ermöglicht es, die gesamten Entwicklungs- und Testaktivitäten direkt im Modell auszuführen. Die abstrakteren Modelle sind dabei oftmals leichter zu lesen als der Code. So wird etwa aus einem Zustandsübergang eine ganze Reihe von Anweisungen generiert, die auch notwendig für die korrekte Funktion sind. Die Modellierung gestaltet sich aber viel einfacher.

Der gleiche Schritt der höheren Abstraktion erfolgte auch beim Übergang von Assembler nach C. Nun lässt sich dieser Vorteil erneut bei der Verlagerung von Entwicklungsaktivitäten ins Modell nutzen. Abstraktion bedeutet Vereinfachung, da dabei bewusst Dinge ausgeblendet werden, die für die Durchführung der eigentlichen Kernaufgabe nicht notwendig sind. Das reduziert einerseits den Aufwand und andererseits Fehlerquellen für die Entwicklung der Software.

Fazit

Softwaregenerierung in C für eingebettete Systeme aus UML-Modellen ist möglich. Durch eine Abbildung der UML-Konzepte in eine funktionsorientierte Welt lassen sich Teile oder ganze Softwareprojekte aus UML generieren. Der dabei entstehende Code bleibt lesbar und ist bei Bedarf auch „manuell“ weiter verwendbar. Entsprechende Werkzeuge sind am Markt verfügbar und lassen sich direkt für die Entwicklung einsetzen (z. B. EnAr Uml2Code).

In Kombination mit SysML im gleichen Modell werden ganze Systemspezifikationen, inklusive Elektronik und Mechanik, nachverfolgbar spezifiziert und dokumentiert, ohne dabei Medienbrüche zu erzeugen oder die grafische Modellierung verlassen zu müssen. Neben der Codegenerierung ist auch die Fehlersuche auf Modellebene möglich. Damit wird letztlich die Entwicklung komplett auf die Modellebene gezogen, was Vereinfachungen und einen intuitiveren Umgang aufgrund der grafischen Darstellung erlaubt. Hier gilt: „Ein Bild sagt mehr als tausend Worte.“ (ane)

Literatur

[1] Oliver Alt; Modellbasierte Systementwicklung mit SysML; Hanser Verlag 2012 (www.sysml-praxis.de)

Dr. Oliver Alt
arbeitet als Senior Consultant bei der LieberLieber Software GmbH in Wien und ist Autor zweier Bücher zum Thema SysML.