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

iX DEVELOPER 1/2016

IoT-Know-how:
Best Practices und Projektberichte

Auf der Heft-DVD

Über 7 GByte für Entwickler

Sponsored Software: Intel Parallel Studio XE 2016

Entwicklungswerkzeuge: Eclipse, Arduino, Atom

Tools, Frameworks, Sprachen und mehr: openHAB, AllJoyn, Kura, Elixir, Lua, Apache-Projekte aus den Bereichen Big und Smart Data sowie JavaScript-Tools für das Internet der Dinge

Entwickeln für das Internet der Dinge

Grundlagen:

Elektronik, Sensorik und Software

Open Source im Internet der Dinge

Know-how:

Nutzerschnittstellen konzeptionieren

IoT-Architektur richtig umsetzen

Sensordaten sammeln und verarbeiten

Kommunikation:

Protokolle harmonisieren

Die wichtigsten Messaging- und Sensorprotokolle im Überblick

Sonderdruck

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

IoT-Projekte sicher planen

4 192045 312906 01

€ 12,90

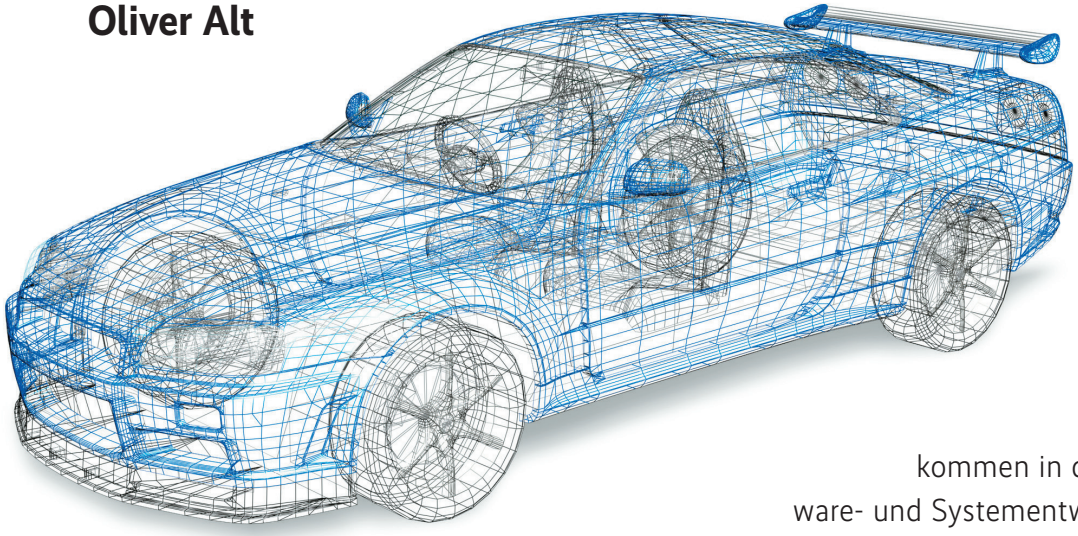
Österreich € 14,20 • Schweiz CHF 25,80
Benelux € 14,80 • Italien € 16,80

www.ix.de

Realisierung komplexer Projekte

Blaupause

Oliver Alt



Modelle kommen in der Software- und Systementwicklung derzeit überwiegend zu Dokumentationszwecken und zur Darstellung der Nachverfolgbarkeit (Traceability) zum Einsatz. Sie zur Softwaregenerierung zu nutzen, kann jedoch gerade für komplexe Systeme Vorteile bieten.

In der Architektur dienen Modelle und Pläne seit jeher dazu, vor der Umsetzung ein Bild des zu erstellenden Bauwerks zu vermitteln. Auch im Software Engineering kommen Modelle zur Verdeutlichung von Sachverhalten zum Einsatz. Ein Beispiel sind die Flussdiagramme, die in den 1960er-Jahren aufkamen. Mit dem Erstarken der Objektorientierung in den 1990ern fassten OMG und ISO einige für Softwaremodelle verwendete Notationen zusammen und standardisierten sie als Unified Modeling Language (UML) [a].

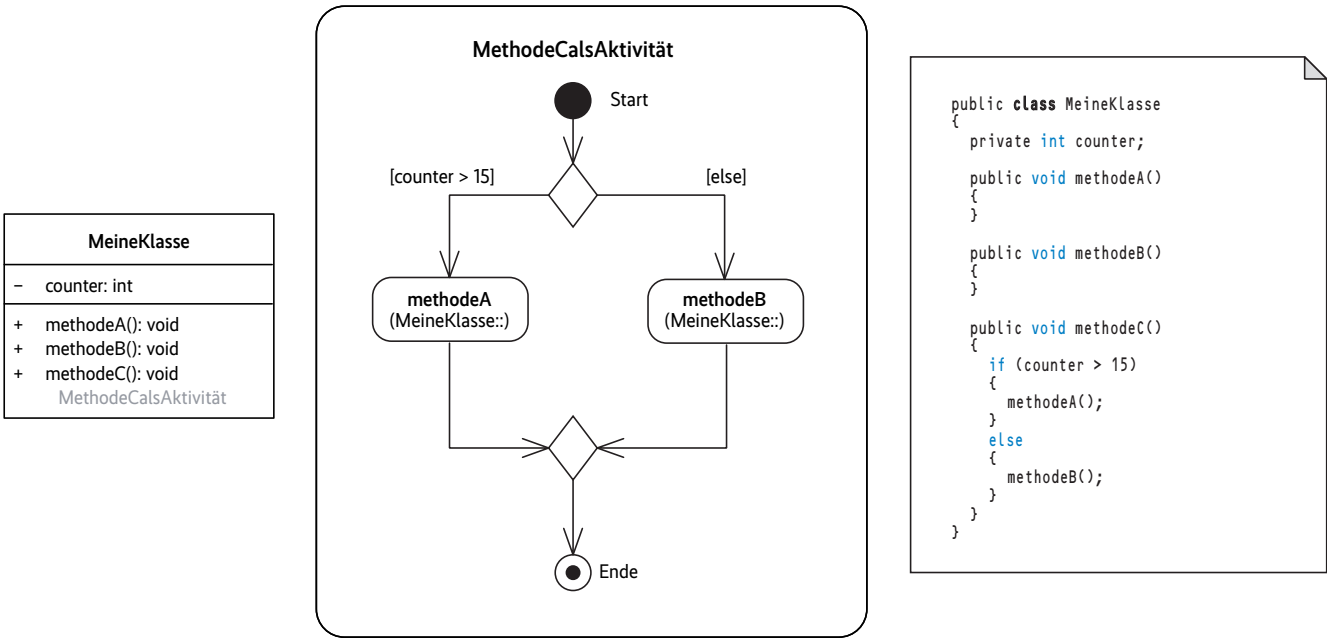
Aus ihren Sprachelementen sind inzwischen viele weitere Modellierungsstandards im Umfeld des Software und Systems Engineering entstanden. Zu nennen sind beispielsweise die Systems Modeling Language (SysML) [b] zur Modellierung von mechatronischen Systemen oder die Business Process Model and Notation (BPMN) zur Beschreibung von Geschäftsprozessen. Sie decken spezielle Einsatzbereiche (Domänen) ab und sind genau genommen nur domänenspezifische Anpassungen der UML-Notationselemente. Man nennt so etwas in der UML-Welt ein UML-Profil. Mithilfe des Profilmechanismus lassen sich neue Sprachelemente aus bestehenden Elementen herleiten.

Abstrakte Darstellung

Zwar gibt es einige Softwareangebote, die eine Umwandlung von UML-Konstrukten in Code versprechen, um befriedigende Ergebnisse zu erzielen, müssen die Modelle jedoch sehr komplex sein. Zu Dokumentationszwecken lassen sie sich eher informativ gestalten, womit sie wiederum nicht ohne Weiteres als Codevorlage dienen können. Eine zusätzliche Hürde ist die häufig fehlende Abstraktion in der modellbasierten Entwicklung. Zu abstrahieren bedeutet in dem Kontext, Elemente so auszublenden, dass das Modell immer noch einen bestimmten Zweck erfüllt, seinen Nutzer aber nicht mit Details überfrachtet. Architekturmodelle etwa stellen Gebäude zwar oft maßstabsgetreu, jedoch nicht fotorealistisch dar. Auch wird meist nur die Geometrie modelliert und keine Details wie Fenster. Sie werden weggelassen (abstrahiert), da Fenster für den Einsatzzweck – sich einen Eindruck der Stadtarchitektur zu verschaffen – nicht notwendig sind.

Um etwas vollständig in einem Modell abzubilden, bedarf es mitunter mehrerer Abstraktionsebenen. Um bei der Architektur zu bleiben: Es gibt spezielle Pläne für unterschiedliche Zwecke. Zum Beispiel braucht ein Elektroinstallateur genaue Angaben, wo welche Leitungen laufen sollen. Für das Unternehmen, das den Rohbau erstellt, sind dagegen die genaue Lage der Wände und Fundamente sowie die einzusetzenden Materialien wichtig. Beide Pläne desselben Gebäudes zeigen so einige Aspekte und abstrahieren von anderen.

In Softwaremodellen, aus denen Code generiert wird, mangelt es oft an der Abstraktion zwischen Quelltext und Modell. In Abbildung 1 sind ein UML-Aktivitätsdiagramm und der da-



Im Idealfall sollte sich aus dem Modell Code erzeugen lassen (Abb. 1).

raus resultierende Code gegenübergestellt. Durch eine fehlende Abstraktionsebene sind oft viele Modellelemente nötig, um den zu generierenden Code zu beschreiben. Folglich stimmt in vielen Fällen das Kosten-Nutzen-Verhältnis beim Einsatz grafischer Modellierung noch nicht. Ein Grund dafür ist, dass die textorientierten Softwareentwicklungsumgebungen in den letzten Jahren durch verbesserte Hilfsmechanismen zunehmend komfortabler geworden sind. IntelliSense, automatische Syntaxprüfung und Vorschläge während des Schreibens erleichtern die Arbeit so, dass der Aufwand der Codeeingabe deutlich geringer scheint. Um die Vorteile der Modellierung und die auf ihr aufbauende Entwicklung für Softwaresysteme zu nutzen, sind folglich besser Modelle zu verwenden, deren Abstraktionsgrad höher liegt als der des daraus generierten Codes.

Modelle für das IoT

Gebäudeautomatisierung und Industrie 4.0 sind nur zwei der vielen Einsatzfelder des Internets der Dinge und erfordern unter anderem durch ihr hohes Maß an Vernetzung geeignete Mittel, um Zusammenhänge im verwendeten System zu beschreiben. Modelle sind in dem Kontext ideal und eine Möglichkeit, um aus ihnen automatisch Code und Konfigurationsdaten zu generieren. Das kann einige Arbeit ersparen.

Um einen ausreichend hohen Grad an Abstraktion zu erreichen, hilft es oft, wenn man domänenspezifisches Wissen in Modelle und Modellierungssprache einbringt. Darunter versteht man Notationen und Eigenheiten aus dem Einsatzbereich der Systeme, die entworfen werden. Beispiele für speziell angepasste Notationen sind technische Zeichnungen, Schaltpläne oder Blockdiagramme aus der Systemtheorie der Regelungstechnik. Damit lässt sich ein System oder eine Software eher logisch statt implementierungsnah beschreiben.

Abbildung 2 zeigt ein Architekturmodell eines IoT-Systems aus dem Bereich Ambient Assisted Living für Sturzerkennung in Kombination mit einem Gebäudeautomationssystem nach dem Konnex-Standard (KNX) [c]. Zu erkennen sind unterschiedliche Hardwarekomponenten wie Sensoren und Aktoren und deren Kommunikationsverbindungen untereinander. Für den Datenaustausch kommen in dem Fall Internetprotokolle zum Einsatz.

Aus einem solchen Modell lassen sich mehrere Schlüsse für die Codegenerierung ziehen. Beispielsweise ist sicherzustellen, dass Knoten, über die Nutzdaten zu versenden sind, entsprechende Software zur Hardwarekommunikation enthalten. Auch enthält das Modell Informationen darüber, welche Hardwaretreiber notwendig sind. Entwickelt man nun vorgefertigte Softwaremodule für die unterschiedlichen Kommunikationsprotokolle und Hardwareplattformen, sind sie nur noch auszuwählen und den Hardwareknoten zuzuordnen. Das Prinzip der vorgefertigten Teile existiert etwa im Maschinenbau seit Jahrzehnten. Maschinen bestehen meist aus normierten Elemente (z. B. DIN-Schrauben), die in einer technischen Zeichnung nicht weiter detailliert zu erklären sind. Es reicht ihre abstrakte Darstellung im Rahmen eines funktionalen Zusammenhangs. Das Abstraktionsmaß steigt folglich durch genormte Komponenten.

Statische und dynamische Beschreibung

In der Softwareentwicklung befindet sich eine derartige Normung erst noch am Anfang. Statt normierter und parametrisierbarer Komponenten, die sich einfach einsetzen lassen, gibt es für die gleiche Aufgabe oftmals mehrere Lösungen. Die konsequente Nutzung wiederverwendbarer Elemente wäre also ein erster Schritt zu einer durchgängigen Softwaregenerierung aus einem Modell. Allerdings ist die Software mit dem bislang gezeigten Vorgehen nur statisch konfiguriert. Was zu einer vollständig funktionierenden Lösung noch fehlt, ist die dynamische Beschreibung durch das Modell, also das Verhalten des Systems. Ein Softwaremodell braucht immer die statische (Architektur-) und die dynamische Verhaltensmodellierung, um daraus komplett lauffähige Software und nicht nur Codeteile ableiten zu können.

Ein höheres Abstraktionsmaß kann man in der dynamischen Modellbildung erreichen, wenn man die funktionalen Aspekte eines Systems herausarbeitet und sie entsprechend nutzt. Bei-

spielsweise haben alle IoT-Anwendungen gewisse Gemeinsamkeiten: Sie müssen Sensorwerte einlesen, Aktoren ansteuern, Daten über Kommunikationskanäle zwischen Knoten verschicken, in unterschiedliche Formate umwandeln und verarbeiten, um daraus Aktionen abzuleiten. Wenn man diese Aktivitäten durch die Modellierungssprache unterstützt, sollten sich die meisten denkbaren Anwendungsszenarien damit beschreiben lassen.

Eine auf dem UML-Standard der Zustandsdiagramme fußende Verhaltensbeschreibung, passend zur zuvor modellierten Architektur, ist in Abbildung 3 zu sehen. Beim Erkennen eines Sturzalarms gibt das System eine Warnung weiter, sollte der Benutzer ihn nicht innerhalb von 30 Sekunden durch das Drücken eines Tasters quittieren, der am Gebäudeautomatisierungssystem angeschlossen ist. Dieses einfach verständliche Verhaltensdiagramm lässt sich gemeinsam mit der statischen Architektur nutzen, um den Code zu generieren, der die vorgefertigten Funktionen in den Softwaremodulen aufruft. Für die Kommunikation der Komponenten ist die tatsächliche Hardwarearchitektur zu berücksichtigen. Die dafür benötigte Logik muss daher Teil eines intelligenten Codegenerators sein. Ebenfalls im Modell abgebildet sind die Zusammenhänge zwischen Struktur und Verhalten des Systems, sodass ein Generator beides berücksichtigen kann.

Bei der Arbeit

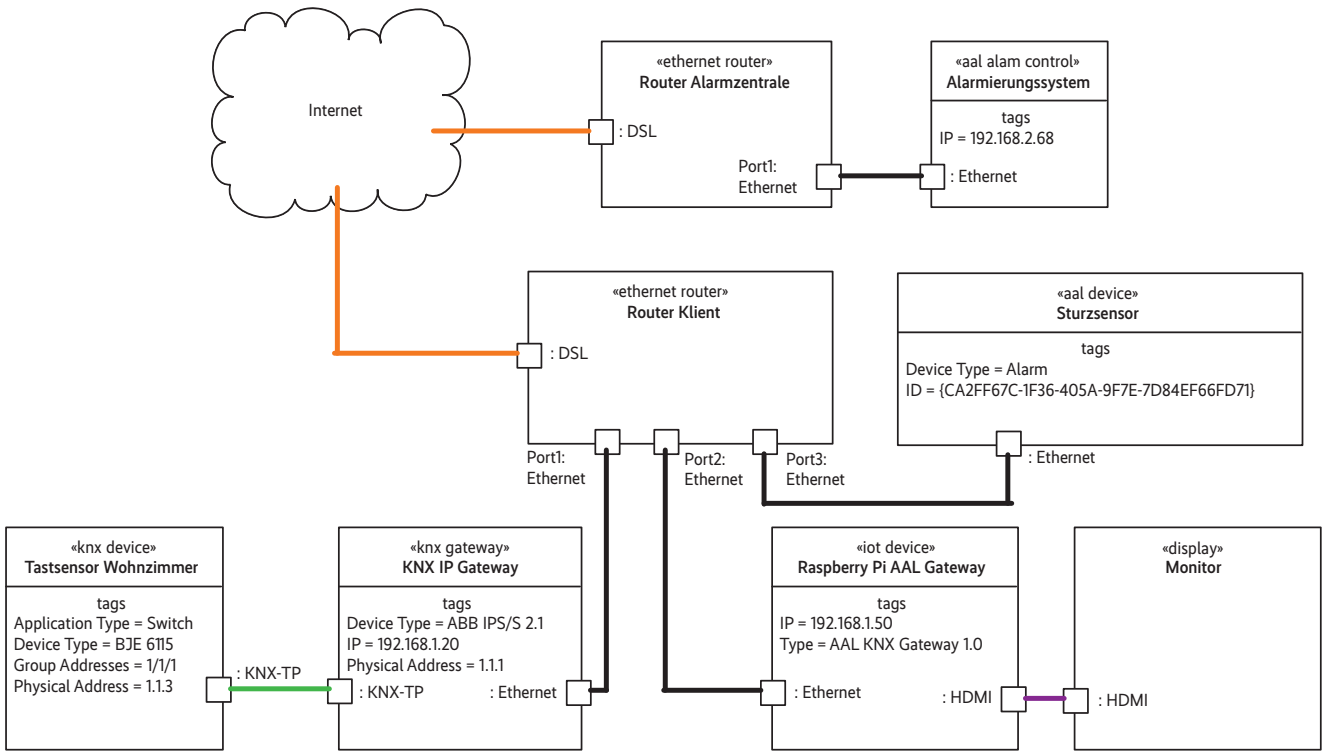
Der Codegenerator analysiert im ersten Schritt das statische Architekturmodell. Aus den dadurch erlangten Erkenntnissen generiert er zunächst einen Rahmen, der die Kommunikation zwischen den technischen Systemen ermöglicht. Im Beispiel ist eine Verbindung zwischen der Gebäudeautomatisierung, die mit dem KNX-Standard arbeitet, und den IP-Techniken notwendig. Dafür muss das System vordefinierte Adapter (KNX/IP-Gateway) entsprechend konfigurieren und Code generieren.

Aus den dynamischen Teilen des Modells entsteht der Verhaltenscode, der dafür sorgt, dass die Betätigung eines Sensors entsprechende Aktionen auslöst. Im Beispiel soll sich ein Sturzalarm durch einen Tastsensor des Gebäudeautomatisierungssystems quittieren lassen. Ihn zu betätigen, bewirkt, dass keine Alarmweiterleitung an eine Zentrale stattfindet (Verhinderung von Fehlalarmen).

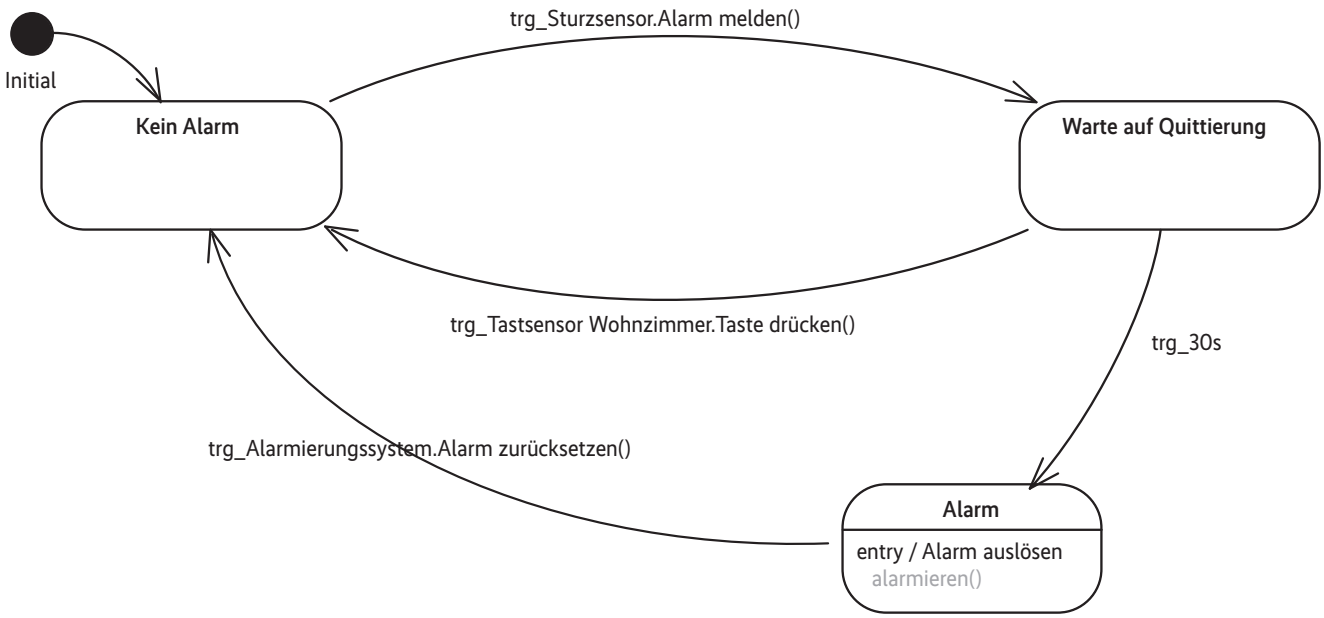
Ein solcher spezialisierter Codegenerator für ein die UML zugrunde legendes Modell lässt sich relativ leicht umsetzen, da moderne Modellierungswerkzeuge wie Enterprise Architect [d] immer eine Datenbank im Hintergrund haben, in der die Modellinhalte hinterlegt sind. Mit Abfragen werden sie für die Codegenerierung eingelesen und passender Quelltext sowie Konfigurationsdaten erzeugt. Das ist der entscheidende Unterschied zwischen Modellierungs- und reinen Grafikwerkzeugen. Bei Ersteren ist immer eine Datenbank (Modell) im Hintergrund. Die grafischen Darstellungen (Diagramme) sind wichtig, um die Datenbank zu füllen, letztendlich entscheidend sind aber die Daten im Modell dahinter. Für das Generieren selbst gibt es auf der Codeseite unterschiedliche Techniken, die über eine direkte Textausgabe hinausgehen. Beispiele sind CodeDom im .NET- und CodeModel im Java-Umfeld [e][f].

OMG-Standards im Einsatz

Statt Codegeneratoren oder Modelltransformationen durch eigene Programme zu realisieren, lässt sich ab und an auch auf entsprechende OMG-Standards zurückgreifen. Bei der Modelltransformationen geht es darum, statt Code zunächst ein weiteres Modell zu generieren. Letzteres hat oftmals einen geringeren Abstraktionsgrad und lässt sich im nächsten Schritt zur Quelltextgenerierung verwenden. Im Gegensatz zu programmierten Generatoren verfolgen die OMG-Standards zu diesen Themen einen etwas anderen Ansatz.



IoT-System aus dem Bereich Ambient Assisted Living (AAL) in Kombination mit einem Gebäudeautomationssystem (Abb. 2)



Ein Zustandsautomat kann das Verhalten des Beispielsystems abbilden (Abb. 3).

Der OMG-Modelltransformationsstandard Query/View/Transformation (QVT) erlaubt es, Regeln dafür zu definieren, wie eine Modelltransformation aussehen soll. Eine QVT-Ausführungsumgebung kann sie einlesen und als Grundlage für Modellumwandlungen verwenden. Der QVT-Standard bietet dabei sowohl eine textuelle als auch eine grafische Notation an, sodass sich Transformationsregeln auch als Modelle erstellen lassen.

Um Code zu erzeugen, hat die OMG den Standard Model To Text (M2T) veröffentlicht. Mithilfe dieser Templatesprache lassen sich statische textuelle Ausgaben mit dynamischen Platzhaltern erstellen, die sich später durch Inhalte aus den Modellen ersetzen lassen. Model To Text ist dabei ähnlich definiert und aufgebaut wie die XML-Transformationssprache XSLT (Extensible Stylesheet Language Transformations). Im Unterschied zur programmierten Modelltransformation oder Codegenerierung, bei der die Regeln von außen nicht direkt sichtbar sind, da sie implizit im Code vorliegen, sind die beiden OMG-Standards sogenannte White-Box-Transformationssprachen.

Zwar arbeiten aktuell einige Stellen an einer vollständigen Implementierung der OMG-Standards (siehe [g]), derzeit ist allerdings noch keine verfügbar. Daher ist man in der Praxis derzeit oftmals noch auf eigene Mittel zur Generierung und Transformation angewiesen.

Fazit

Softwaregenerierung aus Modellen und die modellbasierte Entwicklung funktionieren dann am besten, wenn man die Abstraktion im Hinblick auf den zu generierenden Code entsprechend erhöht und sich außerdem auf standardisierte Komponenten beschränkt. Die Freiheitsgrade der Software werden damit zwar eingeschränkt, jedoch ist genau dieses Prinzip der Standardisierung und Nutzung von vorgefertigten Teilen in vielen anderen technischen Bereichen schon lange Stand der Technik. Das Software- und Systems-Engineering ist im Vergleich dazu noch eine recht junge Disziplin, bei der sich Standards erst noch komplett ausbilden müssen. Wenn das gelingt, sollte auch steigende Komplexität einfach zu handhaben sein und sich IoT-Systeme für die unterschiedlichsten Aufgaben durch Modelle leicht erstellen lassen. Das gilt über technische Systemgrenzen hinweg bis hin zu komplexen Systemen, wie sie für Industrie-4.0-Szenarien typisch sind.

Die Herausforderung für die Zukunft wird sein, Modellierungswerkzeuge und Codegeneratoren in der Handhabung so einfach zu gestalten, dass sie mit heutigen textuellen Softwareentwicklungsumgebungen in der Leistungsfähigkeit und Benutzerfreundlichkeit mithalten können. Weiterhin wird der Umfang der Modelle stetig steigen, sodass sie irgendwann in den Big-Data-Bereich fallen. Eingesetzte Werkzeuge müssen in der Lage sein, auch aus solch großen Modellen in vertretbarer Zeit Code zu erzeugen. Sie entsprechend auszurüsten, ist eine der aktuellen Herausforderungen, denen sich Systementwickler mit Werkzeugen wie Hadoop und MongoDB nähern. (jul)

Onlinequellen

[a]	UML	uml.org
[b]	SysML	omgsysml.org
[c]	KNX	konnex.org
[d]	Enterprise Architect	sparxsystems.com/products/ea/index.html
[e]	CodeDom	msdn.microsoft.com/de-de/library/y2k85ax6%28v=vs.110%29.aspx
[f]	CodeModel	codemodel.java.net
[g]	SysML4Industry	sysml4industry.org

Dr.-Ing. Oliver Alt
arbeitet als Head of Solutions für die LieberLieber Software GmbH in Wien. Er verantwortet alle kundenspezifischen Lösungen im Bereich modellbasierte Software- und Systementwicklung und berät sowie schult regelmäßig Kunden in deren Anwendung.

Alle Links: www.ix.de/ix1517046