

Softwareentwicklung

Plädoyer für Zusammenführung von Ansätzen aus Software Engineering und Regelungstechnik

UML-Modelle können Fahrzeugarchitektur- Rahmen bilden

23. Oktober 2006

Aufgrund von Problemen mit Softwarefehlern im Fahrzeug erwägen einige Autohersteller, die IT-Funktionen wieder zu reduzieren. Eine gründlichere Systemanalyse und der Einsatz von Modellen zur Architekturdefinition könnten die Programmqualität jedoch steigern.

Stolze 43 Prozent der im Betrieb festgestellten Fehler in Steuerungs- und Regelungssoftware sind auf Unzulänglichkeiten in der Analysephase oder der Systemspezifikation zurückzuführen. „Sie wirken sich viel später aus und kosten eine Menge Geld, wenn man sie beheben will“, so Hubert Keller, Projektleiter am Institut für angewandte Informatik des Forschungszentrums Karlsruhe und Präsidiumsmitglied der Gesellschaft für Informatik (GI).

Laut Keller beträgt der Analyseaufwand in einem typischen Softwareprojekt rund sechs Prozent der Gesamtkosten. Investiere man in diesen Bereich das Doppelte – also insgesamt zwölf Prozent –, dann ließen sich auf den Lebenszyklus des Systems gerechnet 20 bis 25 Prozent der Unkosten sparen, da weniger Aufwand für die Fehlerbeseitigung und Wartung zu Buche schlage.

Keller rät daher in dieser frühen Phase zum Einsatz von Softwaremodellen: „Es geht darum, aus der Menge der Anforderungen erst einmal die Kernanforderungen und Inkonsistenzen zu erkennen. Und anhand der Kern-Requirements wird dann eine Systemarchitektur modelliert.“ Werde dieser Prozess vernachlässigt, dann bestehe die Gefahr, dass die Systemarchitektur nicht optimal gewählt werde. „Und diese zu modifizieren, ist sehr aufwändig, kostenintensiv und fehleranfällig.“

Zwei Konzepte konkurrieren

Softwaremodellierung wird im Automobilumfeld schon seit einigen Jahren betrieben. Allerdings konkurrieren im Automobilumfeld zwei Ansätze miteinander.

Der eine wird als modellbasiert – model-based – bezeichnet und stammt originär aus der Regelungstechnik. Der Anwender kann hier regelungstechnische Systeme

wie etwa ein Automatikgetriebe (Bild oben) am PC in einer Experimentierumgebung simulieren. Dieses System wird dann auf einem Echtzeitrechner ausgeführt, um dessen Zeitverhalten zu prüfen und zu verfeinern. Schließlich lässt sich daraus Programmcode generieren.

Das zweite Verfahren nennt sich modellgetrieben oder model-driven. Hierfür kommt die Unified Modeling Language (UML) zum Einsatz, die auf dem Prinzip der Objektorientierung basiert. So werden Programmklassen und ihre Beziehungen sowie die Ablaufstrukturen der Software beschrieben. Über Transformationen vom rein logischen zum plattformspezifischen Modell ist auch hier die Codegenerierung möglich.

Beide Konzepte haben ihre Berechtigung und ihre Grenzen. Ulrich Lefarth, Entwicklungsleiter beim Stuttgarter Tool-Hersteller Etas, räumt ein, dass die „der klassischen Regelungstechnik die Definition von Modulen, Klassen, Prozessen und Messages nicht unbedingt geläufig“ ist. GI-Mann Keller geht noch einen Schritt weiter: Seiner Ansicht ist eine richtige Softwarestruktur mit modellbasierten Werkzeugen „definitiv nicht modellierbar.“ Dagegen sei die UML „ungeeignet, um algorithmenintensive regelungstechnische Funktionen zu beschreiben.“

Keller plädiert dafür, beide Konzepte zu vereinigen. „Das könnte man sich so vorstellen, dass man Softwarearchitekturen und Ablaufstrukturen mit der UML beschreibt und mit Transformatoren umsetzt – und dann Funktionen aus der regelungstechnischen Welt über entsprechende Kennzeichnungen aus Tools wie Matlab generiert und einbindet.“

Dass dies Sinn ergibt, zeigt ein Projekt, das die Fachhochschule Zwickau zusammen mit dem Autozulieferer Bosch absolviert hat. Dabei wurde die Architektur der Steuergerätesoftware mit der UML entworfen.

Ein wichtiges Designkriterium für solche Programme ist die Möglichkeit, Varianten mit unterschiedlichem Funktionsumfang erstellen zu können. Mittels UML profitiert die Software vom objektorientierten Prinzip der Vererbung: Eine Klasse namens Wheel kann eine Basisfunktion wie die Ermittlung der Radumdrehungen pro Minute enthalten. Davon lässt sich eine weitere Klasse ableiten, die neben dieser Funktion weitere Aufgaben erfüllt, wie etwa die Anti-Schlupf-Regelung. fg