Codegenerierung - was man damit (nicht) machen kann
Varianten, Möglichkeiten und Einschränkungen

Horatiu O. Pilsan, Robert Amann, FH Vorarlberg

Die Codegenerierung ist ein wesentlicher Teil eines modellbasierten Entwicklungsprozesses. Der Vortrag vermittelt die wichtigsten Informationen darüber, was man sich von der Codegenerierung in Embedded Systems erwarten soll und was nicht. Ausgehend von der Grundlage, dem (Struktur-, Ablauf-, oder Regler-) Modell, werden die Varianten (signalfluss- und kontrollfluss-, zeit- und ereignisbasiert) in ihren Unterschieden durchleuchtet. Die dafür notwendigen Laufzeitsysteme werden analysiert und deren mögliches Zusammenspiel in einem Gesamtsystem betrachtet.

Der Sinn von Modellierung, Simulation, Codegenerierung

Modellierung ist ein inhärentes Element menschlichen Denkens. Es ist eine Methode, die jeder von uns anwendet, um Komplexität zu handhaben. Lt. Bran Selic ist es „*A human defence mechanism for coping with overwhelming complexity*“(siehe [1]). Wir modellieren alle, ohne uns dessen oftmals bewusst zu werden. Eine explizite Modellierung bedeutet eine formalisierte Darstellung, die ein gemeinsames Verständnis ermöglicht. Das bedeutet, dass Konventionen gefolgt wird, die jenen, die das Modell verwenden, bekannt sind. Ein Beispiel dafür ist die Legende (Konvention) eines Stadtplans (Modell). Daraus entstehen die bekannten Vorteile für die Kommunikation, Dokumentation, Übergabe, Einschulung, etc.

Darüber hinaus ist Modellierung die Grundlage für Simulation, weshalb die Begriffe gern gemeinsam verwendet werden. Auf die Simulation wird hier, trotz ihrer Bedeutung und Verbreitung, nicht im Detail eingegangen.

Aber Modelle sind auch der Ausgangspunkt für Codegenerierung. Deren Nutzen kann im Wesentlichen durch folgende drei Punkte dargestellt werden:

- Der Aufwand und die Zeit für die Implementierung können reduziert werden.
- Modell und Code können (leichter) synchron gehalten werden.
- Die Anzahl Fehler im Code kann verringert werden.

Im Weiteren bezieht sich dieser Artikel nur auf jene Modelle und deren Codegenerierung, bei denen der generierte Code auf dem Embedded System läuft. So wird z.B. nicht auf die Generierung von Code eingegangen, der bei Hardware-in-the-Loop notwendig ist, um die Modelle, die das Umfeld eines Embedded Systems simulieren, auf einem Rechner quasi-zeitkontinuierlich ausführen zu können. Details dazu siehe z.B. [2].

Auch wird auf die große Bedeutung der Verifikation und Validierung der Modelle nicht eingegangen. Einige Prinzipien dazu finden sich z.B. in [3] und [4].

In den folgenden drei Unterkapiteln werden die für die Codegenerierung relevanten Typen von Modellen, der erzeugte Code und etwaige Laufzeitsysteme beschrieben, bevor Gemeinsamkeiten, Unterschiede und Einschränkungen skizziert werden.

Strukturmodelle

Strukturmodelle sind sehr gut geeignet um die Aufteilung (logisch, physikalisch, etc.) einer Software in Elemente und die Beziehungen zwischen diesen darzustellen. Diese Modelle sind statisch, denn sie beschreiben kein Verhalten. Somit werden sie nicht für Simulationszwecke erstellt, sondern dienen primär der Darstellung eines Software-Systems. Für solche Modelle können z.B. einige Diagramme der Unified

Modeling Language (UML) verwendet werden, wobei das Klassendiagramm das am weitesten verbreitete ist.
Zahlreiche Softwarepakete zur Modellierung bieten die Möglichkeit Code aus Strukturmodellen zu erzeugen. Als bekanntestes Beispiel sei hier ebenfalls das UML Klassendiagramm erwähnt. Das Ergebnis ist ein Code-Framework mit Konstruktoren/Destruktoren sowie Methoden/Operationen in der gewählten Programmiersprache, z.B. C++. Der Teil vom Code, der das Verhalten implementiert, kann dann entweder im Modell oder direkt im Code eingefügt werden. Der große Vorteil dieses Vorgehens liegt in der erleichterten Synchronisation zwischen Modell und Code. Viele Werkzeuge unterstützen Roundtrip Engineering, das die Synchronizität sicherstellt. Ein Laufzeitsystem ist in diesem Fall nicht notwendig.

Zeitbasierte Verhaltensmodelle
Die zeitbasierten Verhaltensmodelle haben ihren Ursprung im Begriff „System Dynamics“, der von Prof. Jay Forrester vom Massachusetts Institute of Technology Mitte der 50er Jahre geprägt wurde (siehe [5], S. 38). Wichtig sind die Rückkopplungsschleifen, die das „Herz“ eines jeden Systems bilden. Das Modell ist im Grunde eine Darstellung der daraus resultierenden differential-algebraischen Gleichungssysteme. Somit bedeutet Simulation auch die numerische Lösung dieser Gleichungssysteme. Zeit wird sowohl kontinuierlich modelliert, um das Verhalten der Systeme (z.B. Regelstrecke) zu beschreiben, als auch diskret, für das Verhalten der computerbasierten Kontrolle (z.B. Regler). Da bei solchen Modellen die Verbindungen zwischen den Elementen in Form von Signalen dargestellt werden, wie in Abb. 1 beispielhaft veranschaulicht, kann man hier auch von signalbasierten Systemen sprechen.

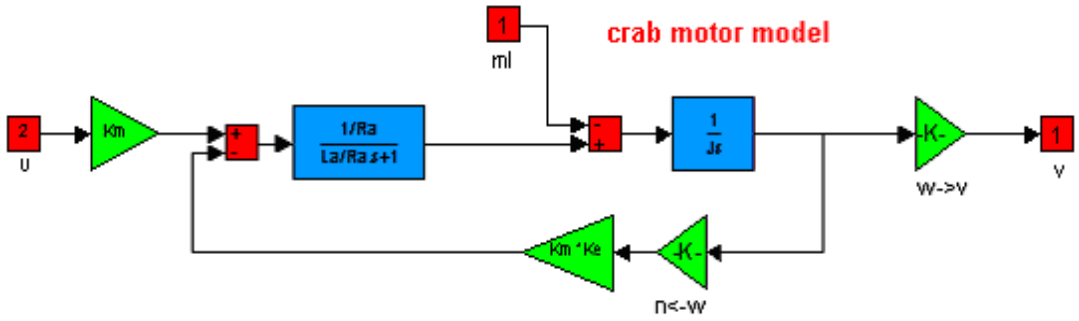

Abb. 1: Simulink-Modell eines DC-Motors

Bei signalbasierten Systemen bedeutet Codegenerierung die Umwandlung der Algorithmen (Beispiel: Regler) in eine Programmiersprache, meistens „C“. Dabei ist spätestens vor der Codegenerierung eine Zeitdiskretisierung dieser Komponenten durchzuführen. Für eine korrekte Ausführung ist es essentiell den richtigen Zeitpunkt einzuhalten. Es wird in der Regel ein zyklisch wiederholendes Verhalten benötigt, das auf einem zeitbasierten Laufzeitsystem aufsetzt. Als weit verbreitetes Beispiel dafür sei hier ein Echtzeitbetriebssystem erwähnt, das präemptives prioritätsbasiertes Scheduling verwendet. Die Prioritäten der Threads werden mittels der Rate-Monotonic-Methode vergeben. Das bedeutet, dass ein Thread eine umso höhere Priorität hat, umso kürzer seine Periode ist. Damit werden schnellere Prozesse

bevorzugt. Meistens ist in diesem Fall die Periode auch gleichzeitig die Deadline, bis zu welcher der Thread abgearbeitet werden muss.

Ereignisbasierte Verhaltensmodelle

Im Gegensatz dazu beschreiben ereignisbasierte Verhaltensmodelle Prozesse, Sequenzen, Abläufe in denen Ereignisse (Events) als Auslöser von Aktionen dienen. In diesem Zusammenhang wird gern der Begriff „Discrete Event Modeling/Simulation“ verwendet. Ein gutes Beispiel hierfür sind Modelle von Zustandsautomaten, die u.a. mit UML-Zustandsdiagrammen dargestellt werden können, wie in Abb. 2 beispielhaft veranschaulicht. Das Verhalten eines Automaten wird dadurch bestimmt, dass als Folge eines Ereignisses eine Transition ausgelöst wird, wodurch sich einerseits der Zustand des Systems ändert und andererseits Aktionen ausgeführt werden. Der Begriff kontrollflussbasierte Systeme ist für solche Modelle ebenso zulässig. Simulation bedeute in diesem Fall, das Verhalten des Systems nach Eintreffen eines Events nachzubilden.

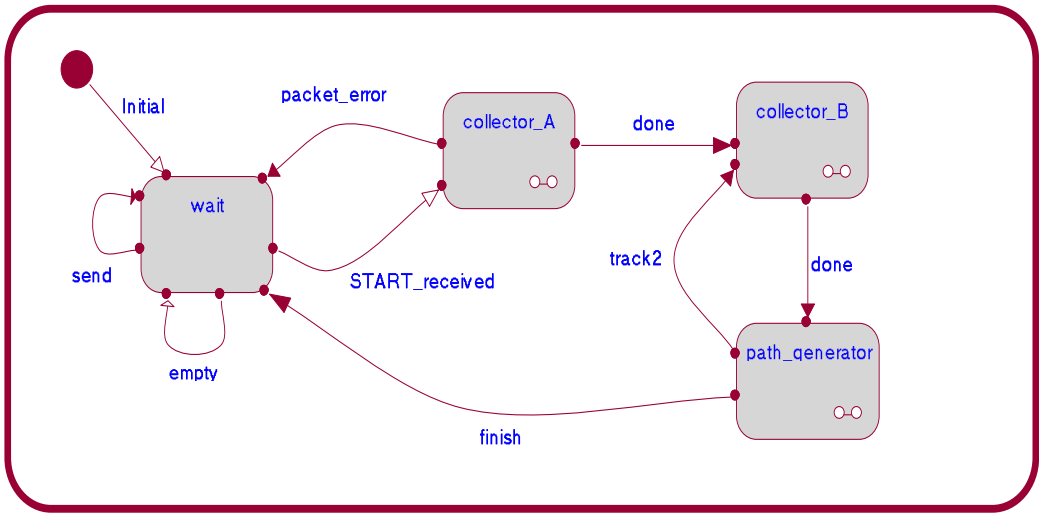

Abb. 2: UML-Zustandsdiagramm

Im Fall ereignisbasierter Systeme spielt das Laufzeitsystem eine wesentliche Rolle. Ein Beispiel dafür ist ein Zustandsautomat, der die Ausführung des Verhaltens ermöglicht, das mit einem UML Zustandsdiagramms modelliert wurde. Dazu wird das in Abb. 3 dargestellte Prinzip verwendet, das einen Event-Dispatcher und eine Event-Queue beinhaltet und dem Run-to-Completion-Paradigma folgt, wie es in der UML-Spezifikation (siehe [3], S. 314f) empfohlen wird: der Dispatcher erlangt erst wieder die Kontrolle über das System, nachdem sämtliche Aktionen der ausgelösten Transition vollständig abgearbeitet wurden. Zeiten sind in diesem Fall zyklische Ereignisse oder Verzögerungen. Sie werden ebenfalls als Ereignisse modelliert und mittels Betriebssystem-Timer implementiert. Das Run-to-Completion-Paradigma hat den Vorteil, dass das Verhalten des Automaten eindeutig bestimmt ist und verhältnismäßig einfach implementiert werden kann. Um Echtzeitprobleme zu meiden, kann es notwendig sein, dass mehrere Automaten in parallel laufenden Threads aktiv sind, wie in Abb. 3 beispielhaft dargestellt.

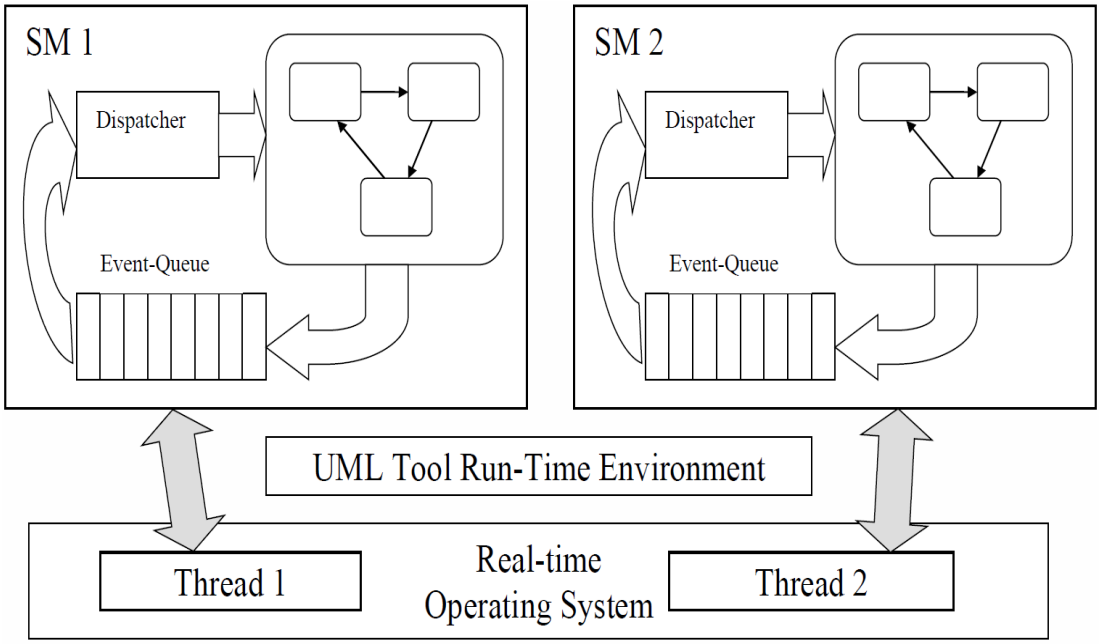

Abb. 3: UML-konforme Implementierung von Zustandsautomaten

Zur Codegenerierung

Codegenerierung hat gewisse Eigenschaften und Einschränkungen, die zu berücksichtigen sind, bzw. es gibt Missverständnisse, die auszuräumen sind.

Es gibt kaum ein Modell, aus dem Code generiert wurde, das lauffähig ist, ohne dass man selbst zusätzlichen Code einfügen muss. Das Nutzverhalten muss irgendwie beschrieben und die Anbindung an die Hardware festgelegt werden. Signalbasierte Systeme können ohne zusätzlichen Code auskommen, da das Verhalten z.B. im Regler festgelegt ist. Wenn dann noch die Ein- und Ausgänge direkt als Blöcke im Modellierungswerkzeug verfügbar sind, ist die Hardwareanbindung auch gegeben. Bei den ereignisbasierten Systemen sind aber die Aktionen, die durchgeführt werden, zu spezifizieren. Das erfolgt in der Regel in einer Hochsprache. Ebendort erfolgt die Kopplung an die Hardware.

Auch bleibt es einem Anwender nicht erspart, das Modell, bzw. den generierten Code mit anderen Komponenten zu koppeln, z.B. Kommunikationsbausteinen. Diese sind quasi immer vorhanden und in den Modellierungswerkzeugen nicht abbildbar.

Die Qualität des generierten Codes ist für dessen Verwendung ausschlaggebend. Dazu zählt in erster Linie die Effizienz. Fragen wie: *„Wie ressourcenverschwenderisch sind Codegeneratoren?“ „Kann man den generierten Code produktiv einsetzen?“ „Werden Fixed- statt Floating-Point-Typen unterstützt?“* lassen sich immer leichter im Sinne des Einsatzes von Codegeneratoren beantworten. Wichtig ist es aber das Gesamtbild zu betrachten, neben dem Code selbst auch die Laufzeitsysteme.

Die Lesbarkeit des generierten Codes ist ebenso ein wichtiges Qualitätsmerkmal. Vor allem dann, wenn das Ergebnis mit anderem Code zusammenspielen muss, oder wenn Codeteile eingefügt werden müssen (was nicht unbedingt empfehlenswert ist).

Die Unterstützung durch Werkzeuge ist nicht durchgängig. Wichtig ist dabei, dass die Tools die Simulation des entsprechenden Verhaltens unterstützen, denn nur dann kann Code-Generierung produktiv im Entwicklungsprozess eingesetzt werden.

UML-Tools unterstützen struktur- und ereignisbasierte Modelle, während zeit- und ereignisbasierte Modelle z.B. von Matlab unterstützt werden.
Die Beschreibung der Laufzeitsysteme lässt erkennen, dass deren Zusammenspiel in einem Embedded System aufgrund der divergierenden Funktionsprinzipien nicht einfach ist. Zwar werden in den Tools oft Blöcke dafür angeboten und das verwendete Echtzeitbetriebssystem ist das gleiche, aber bei der Kopplung ist zu beachten, dass ein Zeitverhalten entsteht, das von der Implementierung der Laufzeitsysteme abhängt.

Fazit

Codegenerierung ist nützlich und sinnvoll, auch wenn nicht uneingeschränkt. Der Sinn dieses Beitrags ist es nicht, jemanden von deren Einsatz abzuhalten, sondern Merkmale aufzuzeigen, die es zu berücksichtigen gilt.
Wegen der Unterschiede zwischen den Varianten, ist es unbedingt notwendig die Anwendung genau zu analysieren, um den optimalen Einsatz von Codegenerierung festzulegen.

Literatur- und Quellenverzeichnis

[1] http://www.modprod.liu.se/MODPROD2011/1.252942/modprod2011-day2-talk1-keynote-Bran-Selic-Abstraction.pdf
[2] Michael Glöckler „Simulation mechatronischer Systeme. Grundlagen und technische Anwendung“ Springer Vieweg, 2014
[3] http://www.informs-sim.org/wsc11papers/016.pdf
[4] http://www.ltas-vis.ulg.ac.be/cmsms/uploads/File/LosAlamos_VerificationValidation.pdf
[5] Andrei Borshchev “The Big Book of Simulation Modeling: Multimethod Modeling with Anylogic 6” Anylogic North America, 2013
[6] http://www.omg.org/spec/UML/2.5/PDF/

Autoren

Horatiu O. Pilsan arbeitete nach seinem Elektronik-Studium 15 Jahre in der Elektronik-Entwicklung, davon ca. 10 Jahre als Hard- und Softwareentwickler, Projekt- und Teamleiter beim Steuerungshersteller Bachmann Electronic, Feldkirch (AT). Seit Ende 2000 ist er an der Fachhochschule Vorarlberg in Dornbirn (AT) als Hochschullehrer für Elektronik und technische Informatik tätig, mit F&E Schwerpunkten in Echtzeitsystemen in Mechatronik und Automatisierungstechnik.

Kontakt

Internet: www.fhv.at
Email: Horatiu.Pilsan@fhv.at

Robert Amann absolvierte 1995 das Telematik Studium an der TU Graz und arbeitete dann vier Jahre als Software- und Hardware-Entwickler für Embedded Systeme bei den Firmen AVL List GmbH und Harris Communications Austria GmbH. Darauf folgten drei Jahre als wissenschaftlicher Mitarbeiter der Fachhochschule Vorarlberg mit den Schwerpunkten Industrielle Kommunikation und Bildverarbeitung. Seit Oktober 2003 arbeitet er als Hochschullehrer für Automatisierungstechnik (Steuerungstechnik, Robotik, Bildverarbeitung) im Automatisierungslabor der Fachhochschule Vorarlberg.

Kontakt
Internet: www.fhv.at
Email: Robert.Amann@fhv.at