Modellierung von deterministischer Software in Simulink

Gerald Stieglbauer, Andreas Werner
Fachbereich Informatik, Software Research Gruppe, Universität Salzburg
{stieglbauer,werner}@SoftwareResearch.net

Abstract: Dieser Beitrag stellt die Integration der Timing Description Language (TDL) in das Modellierungswerkzeug Simulink vor. Das Ziel ist die Etablierung eines Entwicklungsprozesses, der die Realisierung von deterministischer Software für verteilte, eingebettete Systeme signifikant vereinfacht. Dabei wird die textuelle Syntax von TDL durch eine visuell-interaktive Form im Rahmen von Simulink und einer in Simulink eingebundenen TDL Editor Suite besser zugänglich gemacht. Ein Fallbeispiel erklärt die einzelnen Entwicklungsschritte aus der Sicht des Programmierers. Weiters wird die automatische Generierung von Simulink-Modellen unterstützt, welche Zeit- und Funktionsverhalten der entworfenen Programme simulieren. Die Prinzipien von TDL garantieren dem Programmierer, dass Zeit- und Kommunikationsverhalten auf verschiedenen Plattformen identisch sind.

1 Einführung

Die rapide wachsende Komplexität von eingebetteter Software verlangt neue und innovative Methoden, welche ein deterministisches Verhalten der Software sicherstellen. Ein deterministisches Verhalten bedeutet, dass zu einer gegebenen zeitlich geordneten Sequenz von Eingabewerten bei wiederholter Ausführung der Software die gleiche Sequenz von Ausgabedaten (Wertdeterminismus) zu den gleichen Zeitpunkten (Zeitdeterminismus) auftritt [Ki02]. Aufwändige Motorsteuerungen oder X-by-Wire-Systeme machen diese Notwendigkeit besonders in der Automobilindustrie deutlich. Erschwerend kommt hinzu, dass es sich bei modernen eingebetteten Systemen meist um verteilte Systeme handelt, welche nicht-deterministische und schwer beherrschbare Systeme bilden [FHPS04].

Die auf dem momentanen Stand der Technik beruhenden Entwicklungswerkzeuge für verteilte eingebettete Systeme (wie z. B. DaVinci oder dSpace) haben kaum Unterstützung für die Entwicklung von deterministischer, komponierbarer und plattformunabhängiger Software [FHPS04]. Die Time Triggered Architecture (TTA) [Ko97] und FlexRay dagegen basieren auf einer zeitgesteuerten Kommunikationsinfrastruktur, um zeitlich bestimmbares Systemverhalten zu erzielen. Die Sprachen Giotto [HHK01] und dessen Nachfolger, die Timing Description Language (TDL) [Te04], führen eine Abstraktionsebene ein, welche die Software-Entwicklung von der zugrunde liegenden Plattform trennt. Somit verfolgen Giotto und TDL ähnlich wie auch synchrone Sprachen (wie z.B. Esterel [Be00] und Lustre [HCRD91]) das Ziel, Plattformunabhängigkeit durch Einführung von Abstraktionen zu erreichen. Außerdem wurde Giotto und TDL von der grundlegenden Idee zeitgesteuerter

Protokolle (wie sie bei TTA und Flexray zu finden sind) inspiriert. Das von Giotto und TDL eingeführte Prinzip der so genannten FLET (Fixed Logical Execution Time) führt zu Programmen, welche nicht nur bezogen auf das Werte- (Esterel/Lustre) sondern auch auf das Zeitverhalten deterministisch sind [Ki02]. Neben Determinismus und Plattformunabhängigkeit ist die Komposition von Giotto- bzw. TDL-Programmen von großer Bedeutung: Einzelkomponenten lassen sich zu einem größeren Gesamtsystem zusammensetzen, ohne dass dabei die Einzelkomponenten hinsichtlich ihres eigenständigen, deterministischen Verhaltens beeinflusst werden.

In [St03] wurde gezeigt, wie Teile der Sprache Giotto in das weit verbreitete Modellierungswerkzeug Simulink integriert werden können. Als Fortsetzung dieser Arbeit präsentieren wir die vollständige Integration zur Definition und Simulation von Programmen der Sprache TDL in Simulink. Die textuelle Syntax von TDL wird dabei auf visuell-interaktivem Weg für industrielle Anwendungen zugänglich gemacht. Dieser Artikel ist wie folgt strukturiert: Abschnitt 2 skizziert anhand eines Fallbeispiels die einzelnen Schritte des Entwurfs und der Simulation eines TDL-Programms aus Sicht des Entwicklers. Abschnitt 3 behandelt die konkreten Vorteile des hier vorgestellten Software-Entwicklungsprozesses und der zugehörigen Werkzeuge.

2 Entwurf und Simulation eines TDL Programms

Abbildung 1 zeigt die TDL-Werkzeugkette, welche durch horizontale und vertikale Linien unterteilt wird. Die vertikalen Linien verdeutlichen die für TDL typische Trennung

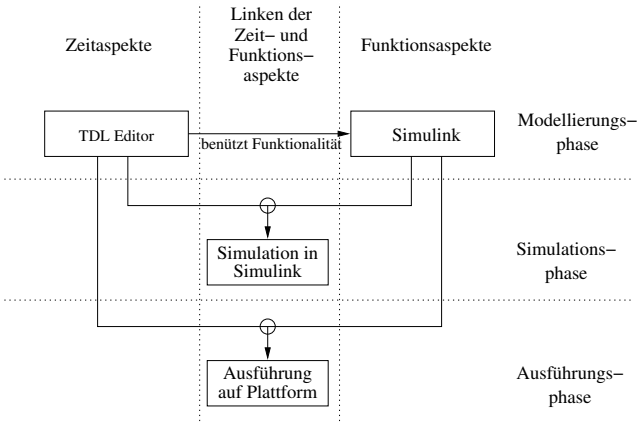

Abbildung 1: Überblick über die verwendete Werkzeugkette

von Zeitverhalten und Funktionalität während des Software-Entwicklungsprozesses. Diese Trennung führt zu einer deutlichen Reduzierung von Komplexität im Vergleich mit anderen Strategien, welche beide Aspekte vermischen oder keine exakte Definition der

Zeitaspekte ermöglichen (wie prioritätsgesteuerte Systeme a lá DaVinci [We02]). Die horizontalen Linien trennen drei verschiedene Phasen der Software-Entwicklung: Die Modellierungs-, Simulations- und Ausführungsphase. Im Folgenden erklären wir die einzelnen Schritte der Modellierungs- und Simulationsphase aus der Sicht des Entwicklers anhand eines Fallbeispiels. Dieses Fallbeispiel befasst sich mit der Ansteuerung einer Drosselklappe, welche in Automobilen zur Steuerung der Luftzufuhr des Motors eingesetzt wird. Die Drosselklappe verfügt über zwei Mess-Sensoren, die unabhängig voneinander den aktuellen Winkel der Regelklappe ermitteln. Zusätzlich zu diesen Werten wird ein dritter Wert, welcher durch einen Sensor am Gaspedal ermittelt wird, dazu verwendet, den gewünschten Winkel der Klappe zu ermitteln. Weisen die beiden ersten Sensoren signifikant unterschiedliche Werte auf, so wird der Regler in einen Fehlermodus übergeführt, der dafür sorgt, dass sich das System in einem für den Anwender sicheren Zustand befindet. Die präsentierten Werkzeuge bieten weiters die Ausgangsbasis für automatische Code-Generierung (Ausführungsphase). Diesen Vorgang hier detailliert zu beschreiben würde den Umfang dieses Artikels sprengen. Eine Beschreibung der Prinzipien findet sich jedoch in [St03].

2.1 Entwurf des Kontrollsystems mit Simulink und der TDL Editor Suite

Regelsystem als Simulink-Modell Im ersten Schritt, der Modellierungsphase, wird ein Simulink-Modell entworfen, welches zwei miteinander verbundene Blöcke enthält, die für die Modellierung eines Regelsystems notwendig sind: Die Regelstrecke und den Regler (TDL-Programm). Während die Regelstrecke durch Standard-Simulink-Blöcke modelliert wird, stellen wir für den Regler einen eigenen Block aus einer TDL-Bibliothek zur Verfügung. Um diesen Block zu konfigurieren, verwendet man die TDL Editor Suite, welche durch einen Doppelklick auf das Blocksymbol gestartet wird. Für den Simulink-Benutzer ist dies analog zu der Verwendung eines StateFlow-Blocks in einem Modell.

TDL Editor Suite Die TDL Editor Suite umfasst im Wesentlichen drei Editoren, welche die einzelnen Aspekte eines TDL-Programms trennen:

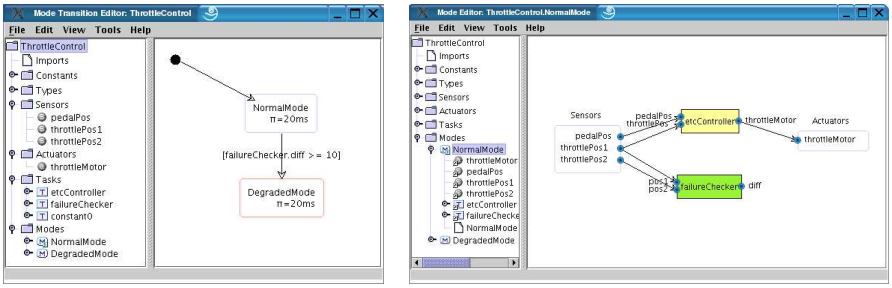

Abbildung 2: Mode Transition Editor (links) und Mode Editor (rechts)

Mode Transition Editor Ein TDL-Programm-Modul besteht aus einer Menge von Modes. Der Mode Transition Editor (s. Abbildung 2, links) gibt einen Überblick über diese Modes und deren Zustandsübergänge (Mode Switches = Mode-Wechsel). Das System befindet sich nach dem Start im Mode `NormalMode`, welche die Drosselklappenposition in Abhängigkeit von der Stellung des Pedals kontrolliert. Weichen die beiden eingelesenen Sensorwerte zu stark voneinander ab, so wird die Bedingung `[failureCecker.diff >= 10]` wahr und das System geht in `DegradedMode` über (Mode Switch).

Mode Editor Durch einen Doppelklick auf ein Mode-Symbol gelangt man zur Definition eines Modes, welche im Wesentlichen aus der Zuordnung von Tasks[1] sowie deren Konfiguration (Task-Frequenz, Task-Kommunikation, etc.) besteht (s. Abbildung 2, rechts). In `NormalMode` werden zwei Tasks ausgeführt: `etcController` berechnet den neuen Ansteuerungswert für den Drosselklappenmotor aus der Stellung des Pedals und der aktuellen Position der Drosselklappe. Die Task `failureChecker` berechnet die Differenz zweier unabhängiger Messungen der Drosselklappenposition. `DegradedMode` (ohne Abbildung) enthält nur eine Task: `constant0` gibt immer den Wert 0 aus, um die Drosselklappe in einer sicheren Position zu halten.

Mode Communication Editor Schließlich müssen wir angeben, welche Werte nach einem Mode-Wechsel in den Zielmode übernommen werden. Durch einen Doppelklick auf jenes Pfeilsymbol, welches im Mode Transition Editor den Übergang zwischen zwei Modes symbolisiert, gelangt man in den Mode Communication Editor (ohne Abbildung). Mit diesem Editor kann man die Werteübergabe zwischen Modes definieren.

Funktionalität in Simulink Die nahtlose Integration der TDL Editor Suite ermöglicht auch den Weg zurück zu Simulink: Ein Doppelklick auf ein Task-Symbol im Mode Editor öffnet den Task-Inhalt, nämlich dessen Funktionalität in Form eines Editor-Fensters in Simulink. Laut Definition unserer Werkzeugkette wird die Funktionalität in Simulink, klar getrennt von den Zeitdefinitionen eines TDL-Programms, unter Verwendung von Standard-Blöcken modelliert.

2.2 Erstellung eines simulierbaren TDL-Modells in Simulink

Nach der Definition von Regelstrecke, TDL-Programm und Funktionalität der TDL-Tasks ist der nächste Schritt in der Werkzeugkette die Simulation des Gesamtsystems. Dafür wird ein Simulink-Modell generiert, welches das Zeitverhalten des TDL-Programms modelliert, in das Regelstrecke und Task-Funktionalität eingebettet sind. Die TDL Editor Suite kann durch Auswahl eines entsprechenden Menüeintrages dieses Modell automatisch im Hintergrund generieren und die Simulation starten. Der Programmierer wiederum kann anhand der Simulationsergebnisse notwendige Modifikationen am TDL-Programm (durch die TDL Editor Suite) oder der Task-Funktionalität (in Simulink) vornehmen.

[1] Tasks können in mehreren Modes wiederverwendet werden.

3 Resultate und Schlussfolgerungen

Mit den präsentierten Werkzeugen lässt sich deterministische Software für verteilte Steuerungssysteme durch Integration der Sprache TDL in das Modellierungswerkzeug Simulink definieren und simulieren. Da Simulink de facto das Standard-Modellierungstool für Regelungstechniker ist, wurde eine möglichst gute Integration von TDL und Simulink angestrebt. Durch die Integration der TDL Editor Suite in Simulink werden Änderungen automatisch zwischen den verschiedenen Editoren propagiert. Die Sprache TDL umfasst wie Giotto das Konzept eines deterministischen und dynamischen Mode-Wechsels. Dieser Mode-Wechsel ist in Simulink nur unter großem Aufwand zu modellieren und wurde im Rahmen des MoDECS-Forschungsprojektes (siehe www.modecs.cc) in Simulink integriert. Die einfache Definition von TDL-Programmen durch die TDL Editor Suite und die automatische Generierung von Simulink-Modellen befreit den Anwender jedoch vom Entwurf und dem Editieren derartiger Modelle. Aufgrund der von TDL eingeführten Abstraktionsebene muss der Programmierer bei der Programmdefinition und der anschließenden Simulation keine Rücksicht auf die Plattform und deren Topologie nehmen: Das identische Laufzeitverhalten auf beliebigen Plattformen hinsichtlich des Simulationsergebnisses wird garantiert. Dies führt zu einer deutlichen Komplexitätsreduzierung bei der Software-Entwicklung. Da die präsentierten Werkzeuge die Ausgangsbasis zur automatischen Code-Generierung bilden, wird dem Anwender somit eine vom plattformunabhängigen Modellentwurf bis hin zur lauffähigen Anwendung durchgängige Werkzeugkette geboten.

Literatur

[Be00] Berry, G.: *The foundations of Esterel*. MIT Press. 2000.

[FHPS04] Farcas, C., Holzmann, M., Pletzer, H., und Stieglbauer, G.: The TDL Advantage. Technical report. Software Research Lab, University of Salzburg, Austria. http://www.SoftwareResearch.net/site/publications/C058.pdf. April 2004.

[HCRD91] Halbwachs, N., Caspi, P., Raymond, P., und D., P.: The synchronous dataflow programming language Lustre. *Proc. of the IEEE, 79(9)*. 1991.

[HHK01] Henzinger, T. A., Horowitz, B., und Kirsch, C. M.: Giotto: A Time-Triggered Language for Embedded Programming. *Lecture Notes in Computer Science*. 2211:166–184. 2001.

[Ki02] Kirsch, C. M.: Principles of Real-Time Programming. *LNCS*. 2491. 2002.

[Ko97] Kopetz, H.: *Real-time Systems: Design Programming for Distributed Embedded Applications*. Kluwer. 1997.

[St03] Stieglbauer, G.: Model-based Development of Embedded Control Systems with Giotto and Simulink. Master's thesis. University of Salzburg. April 2003.

[Te04] Templ, J.: TDL Specification and Report. Technical report. Department of Computer Science, University of Salzburg, Austria. http://www.SoftwareResearch.net/site/publications/C059.pdf. March 2004.

[We02] Wernicke, M.: Design mit System: Funktionsorientierte Entwicklung von verteilter Kfz-Software. *Elektronik Automotive*. December 2002.