

Kapitel 2

Modellierung von Echtzeitsystemen und Werkzeuge

Inhalt

- Motivation
- Grundsätzlicher Aufbau, „Modelle“ und „Models of Computation“
 - Werkzeug Ptolemy
- Synchroner Sprachen (Esterel, Lustre)
 - Reaktive Systeme: Werkzeuge Esterel Studio, SCADE
 - Synchroner Datenfluss: EasyLab
- Zeitgesteuerte Systeme
 - Werkzeug Giotto
- Domänenspezifische Codegeneratoren
 - Werkzeug FTOS
- Verifikation durch den Einsatz formaler Methoden

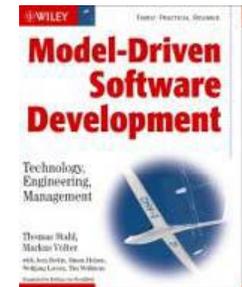
Fokus dieses Kapitels

- Konzepte und Werkzeuge zur Modellierung **und** Generierung von Code für Echtzeit- und eingebettete Systeme
- Voraussetzungen an Werkzeuge für ganzheitlichen Ansatz:
 - Explizite Modellierung des zeitlichen Verhaltens (z.B. Fristen)
 - Modellierung von Hardware und Software
 - Eindeutige Semantik der Modelle
 - Berücksichtigung von nicht-funktionalen Aspekten (z.B. Zeit*, Zuverlässigkeit, Sicherheit)
- Ansatz zur Realisierung:
 - Schaffung von domänenspezifischen Werkzeugen (Matlab/Simulink, Labview, SCADE werden überwiegend von spezifischen Entwicklergruppen benutzt)
 - Einfache Erweiterbarkeit der Codegeneratoren oder Verwendung von virtuellen Maschinen / Middleware-Ansätzen

* Zeit wird zumeist als nicht-funktionale Eigenschaft betrachtet, in Echtzeitsystemen ist Zeit jedoch als funktionale Eigenschaft anzusehen (siehe z.B. Edward Lee: Time is a Resource, and Other Stories, May 2008) http://chess.eecs.berkeley.edu/pubs/426/Lee_TimeIsNotAResource.pdf

Literatur

- Sastry et al: Scanning the issue – special issue on modeling and design of embedded software, Proceedings of the IEEE, vol.91, no.1, pp. 3-10, Jan 2003
- Thomas Stahl, Markus Völter: Model-Driven Software Development, Wiley, 2006
- Ptolemy: Software und Dokumentation
<http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm>
- Benveniste et al.: The Synchronous Languages, 12 Years Later, Proceedings of the IEEE, vol.91, no.1, pp. 64-83, Jan 2003
- Diverse Texte zu Esterel, Lustre, Safe State Machines:
<http://www.esterel-technologies.com/technology/scientific-papers/>
- David Harrel, Statecharts: A Visual Formalism For Complex Systems, 1987
- Henzinger et al.: Giotto: A time-triggered language für embedded programming, Proceedings of the IEEE, vol.91, no.1, pp. 84-99, Jan 2003



Hinweis: Veröffentlichungen von IEEE, Springer, ACM können Sie kostenfrei herunterladen, wenn Sie den Proxy der TUM Informatik benutzen (proxy.in.tum.de)

Begriff: Modell

- Brockhaus:
Ein **Abbild** der Natur unter der Hervorhebung für **wesentlich** erachteter **Eigenschaften** und Außerachtlassen als nebensächlich angesehener Aspekte. Ein M. in diesem Sinne ist ein Mittel zur Beschreibung der erfahrenen Realität, zur Bildung von Begriffen der Wirklichkeit und Grundlage von Voraussagen über künftiges Verhalten des erfassten Erfahrungsbereichs. Es ist um so realistischer oder wirklichkeitsnäher, je konsistenter es den von ihm umfassten Erfahrungsbereich zu deuten gestattet und je genauer seine Vorhersagen zutreffen; es ist um so **mächtiger**, je **größer** der von ihm beschriebene **Erfahrungsbereich** ist.
- Wikipedia:
Von einem **Modell** spricht man oftmals als Gegenstand wissenschaftlicher Methodik und meint damit, dass eine zu untersuchende Realität durch bestimmte Erklärungsgrößen im Rahmen einer wissenschaftlich handhabbaren Theorie abgebildet wird.

Modellbasierte Entwicklung

- Wir beobachten im Bereich eingebettete Systeme seit längerem einen Übergang von der klassischen Programmentwicklung zur einer Vorgehensweise, bei der Modelle (für physikalische Prozesse, für die Hardware eines Systems, für Verhalten eines Kommunikations-netzes, usw.) eine zentrale Rolle spielen
- Modelle werden typischerweise durch verknüpfte grafische Notationselemente dargestellt (bzw. definiert)
- **Funktions-Modelle** sind dabei (meist Rechner-ausführbare) Beschreibungen der zu realisierenden Algorithmen
- In unterschiedlichen Phasen der Systementwicklung kann ein Modell unterschiedliche Rollen annehmen – und etwa zum Funktionsdesign, zur Simulation, zur Codegenerierung verwendet werden
- Im günstigsten Fall kann ein System grafisch notiert („zusammengeklickt“) werden und unmittelbar Code erzeugt werden, der dann auf einem Zielsystem zur Ausführung gebracht wird (Beispiel: EasyKit)
- Diese Entwicklung steht im Einklang mit der generellen Geschichte der Informatik, die immer mächtigere (Programmier-) Werkzeuge auf immer abstrakterem Niveau geschaffen hat

Beispiel FTOS: Modellbasierte Entwicklung fehlertoleranter Echtzeitsysteme

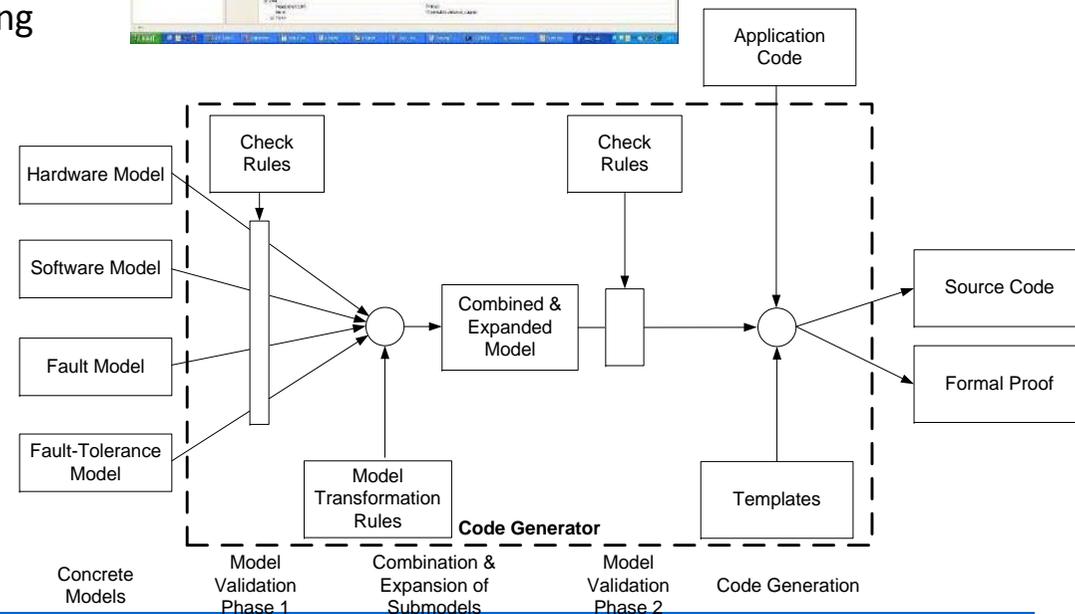
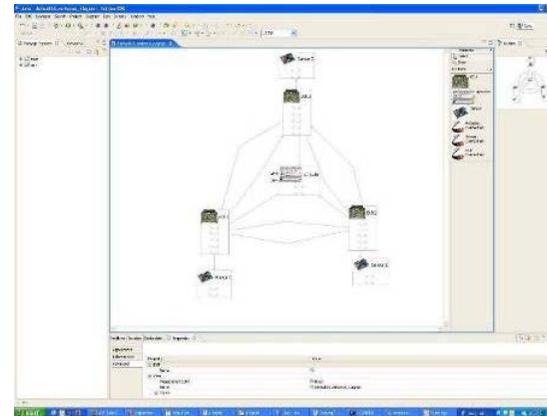
Ziele:

Umfangreiche Generierung von Code auf Systemebene:

- Fehlertoleranzmechanismen
- Prozessmanagement, Scheduling
- Kommunikation

Erweiterbarkeit der Codegenerierung durch Verwendung eines vorlagenbasierten Codegenerators

Zertifizierung des Codegenerators



Vorteile Modellbasierter Entwicklung

- Für die modellbasierte Entwicklung sprechen diverse Gründe:
 - Modelle sind häufig einfacher zu verstehen als der Programmcode (graphische Darstellung, Erhöhung des Abstraktionslevels)
 - Vorwissen ist zum Verständnis der Modelle häufig nicht notwendig:
 - Experten unterschiedlicher Disziplinen können sich verständigen
 - Systeme können vorab simuliert werden. Hierdurch können Designentscheidungen vorab evaluiert werden und späte Systemänderungen minimiert werden.
 - Es existieren Werkzeuge um Code automatisch aus Modellen zu generieren:
 - Programmierung wird stark erleichtert
 - Ziel: umfassende Codegenerierung (Entwicklung konzentriert sich ausschließlich auf Modelle)
 - Mittels formaler Methoden kann
 - die Umsetzung der Modelle in Code getestet werden
 - das Modell auf gewisse Eigenschaften hin überprüft werden

Modellbasierte Ansätze sind erfolgreich ...

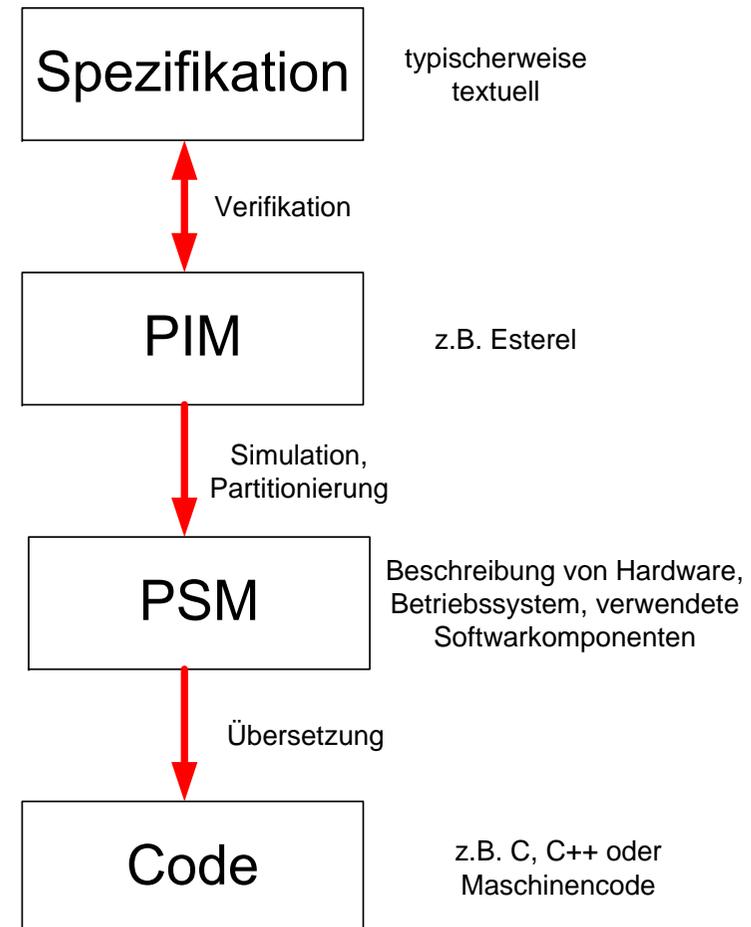
... sie haben unter anderem zu folgendem beigetragen:

- Ablösen des klassischen „Komponenten-Kopie-Ansatzes“ durch das Prinzip der **Klasse-Instanz-Beziehung**
- Definition von **domänenspezifische Architekturmodellen**: Schichtenmodelle, Bushierarchien
- Konstruktion von **hybriden Umgebungsmodellen**: Abstraktere und erweiterte Modelle der Umgebung
- Definition von Entwurfsebenen: Funktionsarchitektur, logische Architektur, technische Architektur (analog zur Strukturierung der Hardware-Entwicklungsschritte: Architektur – Implementierung – Realisierung, (Gene Amdahl, IBM, 1964), siehe VL-homepage)
- Vermittlung von Modellen und Techniken zur Unterstützung der Anforderungsanalyse und -definition
- Vermittlung von Modellen und Techniken zur Beschreibung von Varianten und Produktlinien

(siehe <http://www4.informatik.tu-muenchen.de/~schaetz/MBEES/index.html>)

Beispiel OMG: Model-Driven Architecture (MDA)

- Die Entwicklung des Systems erfolgt in diversen Schritten:
 - textuelle Spezifikation
 - PIM: platform independent model
 - PSM: platform specific model
 - Code: Maschinencode bzw. Quellcode
- Aus der Spezifikation erstellt der Entwickler das plattformunabhängige Modell
- Hoffnung: weitgehende Automatisierung der Transformationen PIM → PSM → Code (Entwickler muss nur noch notwendige Informationen in Bezug auf die Plattform geben)
- <http://www.omg.org/mda>



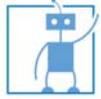
MDA im Kontext von Echtzeitsystemen

- In Echtzeitsystemen / eingebetteten Systemen ist bei einem umfassenden Ansatz ein Hardwaremodell (z.B. Rechner im verteilten System, Topologie) schon in frühen Phasen (PIM) notwendig
- Das plattformspezifische Modell (PSM) erweitert das Hardware- & Softwaremodell um Implementierungskonzepte, z.B.
 - Implementierung als Funktion/Thread/Prozess
 - Prozesssynchronisation

Anwendungsbeispiele

- Zur Illustration diverser Konzepte werden in der Vorlesung / Übung mehrere Beispiele verwendet
- Beispiel 1: Aufzugssteuerung
 - Verteiltes System:
 - Steuerungsrechner (einfach oder redundant ausgelegt)
 - Microcontroller zur Steuerung der Sensorik / Aktorik
 - CAN-Bus zur Kommunikation
- Beispiel 2: Ampel





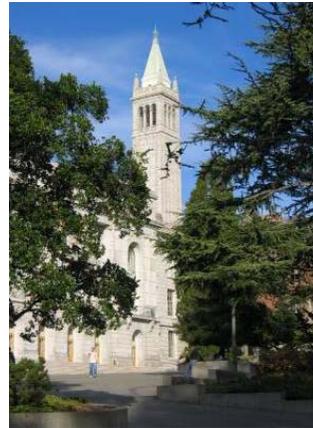
Modellierung von Echtzeitsystemen

Aktoren, Ausführungsmodelle

Werkzeuge: Ptolemy

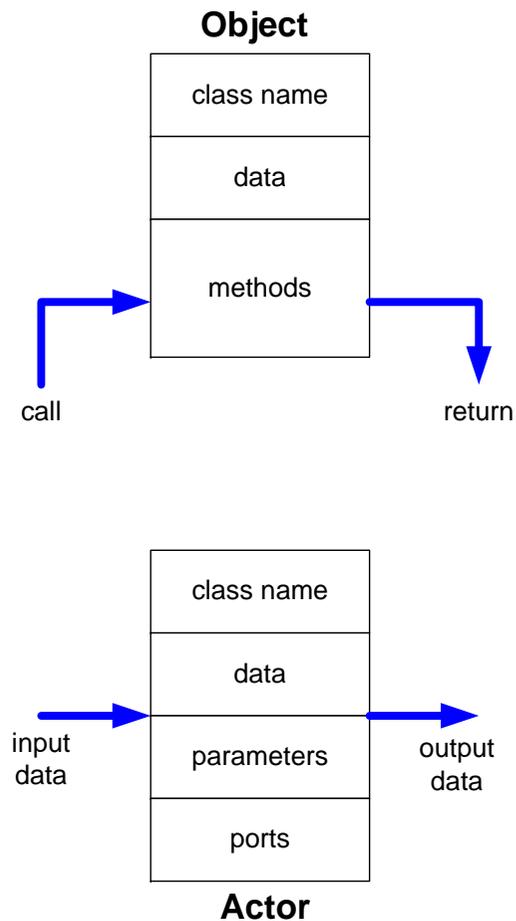
Ptolemy

- Das Ptolemy*-Projekt an der UC Berkeley untersucht verschiedene Modellierungsmethodiken für eingebettete Systeme mit einem Fokus auf verschiedene Ausführungsmodelle (Models of Computation)
- Ptolemy unterstützt
 - Modellierung
 - Simulation
 - Codegenerierung
 - Formale Verifikation (teilweise)
- Weitere Informationen unter: <http://ptolemy.eecs.berkeley.edu/>



***Claudius Ptolemaeus**, (* um 100, vermutlich in Ptolemais Hermii, Ägypten; † um 175, vermutlich in Alexandria), war ein griechischer Mathematiker, Geograph, Astronom, Astrologe, Musiktheoretiker und Philosoph. Ptolemäus schrieb die *Mathematike Syntaxis* („mathematische Zusammenstellung“), später *Megiste Syntaxis* („größte Zusammenstellung“), heute *Almagest* (abgeleitet vom Arabischen *al-Majisṭ*) genannte Abhandlung zur Mathematik und Astronomie in 13 Büchern. Sie war bis zum Ende des Mittelalters ein Standardwerk der Astronomie und enthielt neben einem ausführlichen Sternenkatalog eine Verfeinerung des von Hipparchos von Nicäa vorgeschlagenen geozentrischen Weltbildes, das später nach ihm *Ptolemäisches Weltbild* genannt wurde. (Wikipedia)

Ptolemy: Aktororientiertes Design



- Ptolemy-Modelle basieren auf Aktoren anstelle von Objekten
- Objekte:
 - Fokus liegt auf Kontrollfluss
 - Objekte werden manipuliert
- Aktoren
 - Fokus liegt auf Datenfluss
 - Aktoren manipulieren das System
- Vorteil beider Ansätze: erhöhte Wiederverwendbarkeit
- Vorteil von Aktoren: leichtere Darstellung von Parallelität

Ptolemy: Aktororientiertes Design

- Ausführungsmodelle (models of computation) bestimmen die Interaktion von Komponenten/Aktoren
- Die Eignung eines Ausführungsmodells hängt von der Anwendungsdomäne, aber auch der verwendeten Hardware, ab
- In Ptolemy wird durch die Einführung von „Dirigenten“ (director) die funktionale Ausführung (Verschaltung der Aktoren) von der zeitlichen Ausführung (Abbildung im Direktor) getrennt.
- Aktoren können unter verschiedenen Ausführungsmodellen verwendet werden (z.B. synchron, asynchron)
- Verschiedene Ausführungsmodelle können hierarchisch geschachtelt werden (modal models).
 - Typisches Beispiel: Synchroner Datenfluss und Zustandsautomaten

Example Ptolemy Model of Computation: Synchronuous Dataflow

- Prinzip:
 - Annahme: unendlich schnelle Maschine
 - Daten werden zyklisch verarbeitet
 - Pro Runde wird genau einmal der Datenfluss ausgeführt
- Vorteile:
 - Statische Speicherallokation
 - Statischer Schedule berechenbar
 - Verklemmungen detektierbar
 - Laufzeit kann bestimmt werden
- Werkzeuge:
 - Matlab
 - Labview
 - **EasyLab**

