Kapitel 2

Modellierung von Echtzeitsystemen und Werkzeuge

Inhalt

- Motivation
- Grundsätzlicher Aufbau, Models of Computation
  - Werkzeug Ptolemy
- Synchrone 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

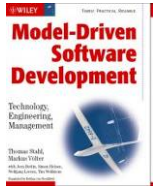
- Konzepte und Werkzeuge zur Modellierung **und** Generierung von Code für Echtzeitsysteme / eingebettete Systeme
- Voraussetzungen an Werkzeuge für ganzheitlichen Ansatz:
  - Explizite Modellierung des zeitlichen Verhaltens (z.B. Fristen)
  - Modellierung von Hardware und Software
  - Eindeutige Semantik 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 betrachet, in Echtzeitsystemen ist Zeit jedoch als funktionale Eigenschaft anzusehen (siehe z.B. Edward Lee: Time is a Resource, and Other Stories, May 2008)

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

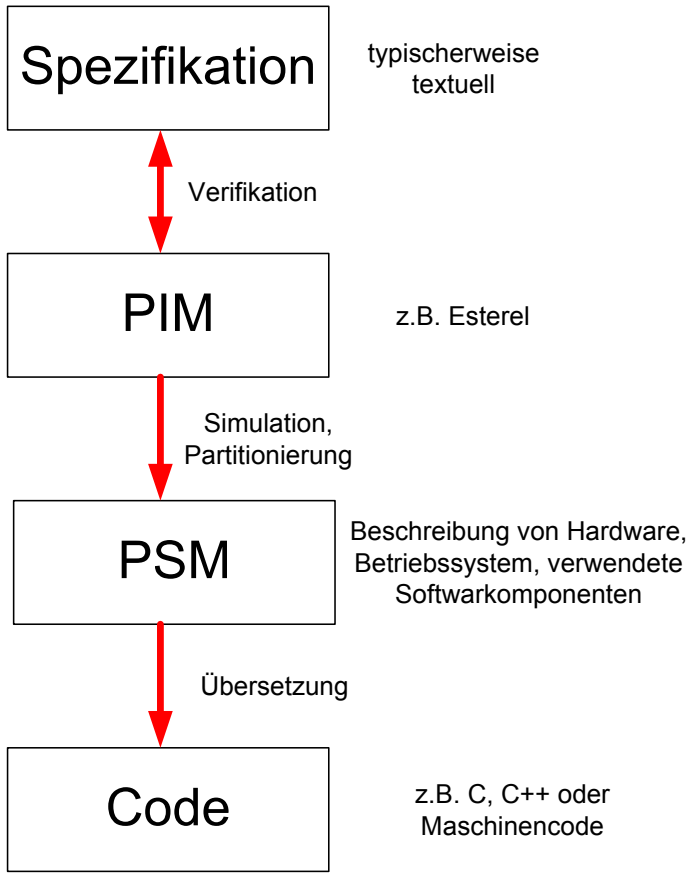
- Brockhaus:
  Ein Abbild der Natur unter der Hervorhebung für wesentlich erachteter Eigenschaften und Außerachtlassen als nebensächlich angesehener Aspekte. Ein M. in diesem Sinne ist ein Mittel zur Beschreibung der erfahrenen Realität, zur Bildung von Begriffen der Wirklichkeit und Grundlage von Voraussagen 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.

Modellbasierte 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 Modell in Code getestet werden
    - das Modell auf gewisse Eigenschaften hin überprüft werden

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
- Hoffung: weitgehende Automatisierung der Transformationen PIM → PSM → Code (Entwickler muss nur noch notwendige Informationen in Bezug auf die Plattform geben)



MDA im Kontext von Echtzeitsystemen

- In Echtzeitsystemen / eingebetteten Systemen ist bei einem umfassenden Ansatz ein Hardwaremodell (z.B. Rechner im verteilten Systen, 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 wir in der Vorlesung / Übung mehrere Beispiele verwenden.
- Beispiel 1: Aufzugssteuerung
  - Verteiltes System:
    - Steuerungsrechner (einfach oder redundant ausgelegt)
    - Microcontroller zur Steuerung der Sensorik / Aktorik
    - CAN-Bus zur Kommunikation
- Beispiel 2: Ampel

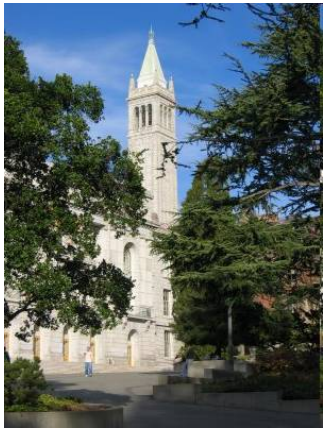
Modellierung von Echtzeitsystemen

Aktoren, Ausführungsmodelle

Werkzeuge: Ptolemy

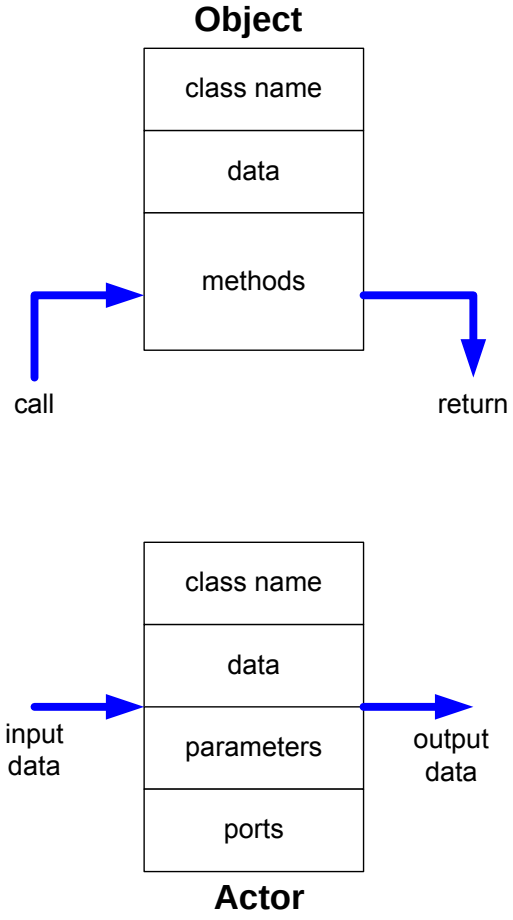
Ptolemy

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



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



Models of Computation

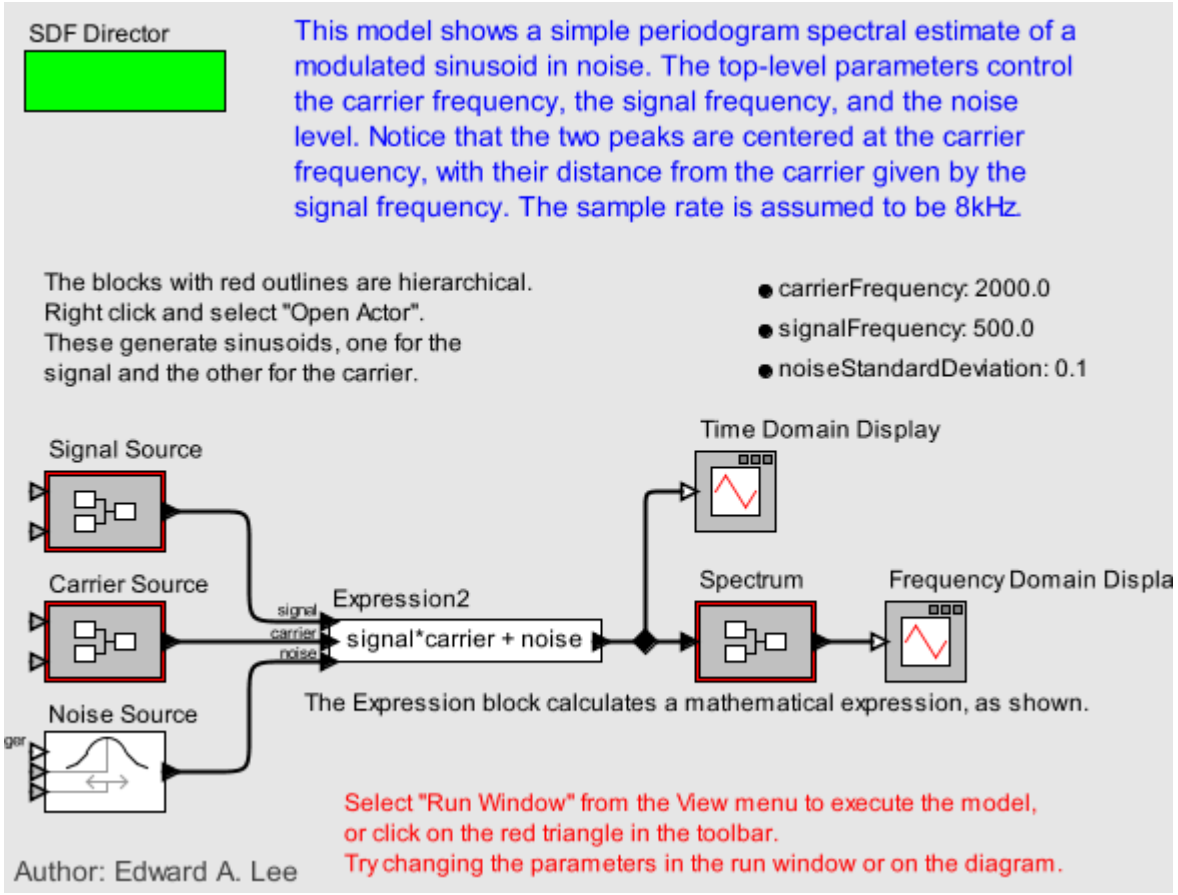
- Ausführungsmodelle (models of computation) bestimmen die Interaktion von Komponenten/Aktoren.
- Die Eignung eines Ausführungsmodells hängt von der Anwendungsdomäne, aber auch der verwendeten Hardware, ab.
- In Ptolemy wird durch die Einführung von Direktoren (director) die funktionale Ausführung (Verschaltung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

Probleme mit Zeit und Daten

Software Zeit: Programmiersprachen unterstützen häufig keine Zeit Daten: sind digital
Digitale synchrone Hardware Zeit: typischerweise zyklische Abarbeitung Daten: sind digital
Digitale asynchrone Hardware Zeit: physikalische Zeit Daten: sind digital
Umgebung Zeit: physikalische Zeit Daten: analog/kontinuierlich

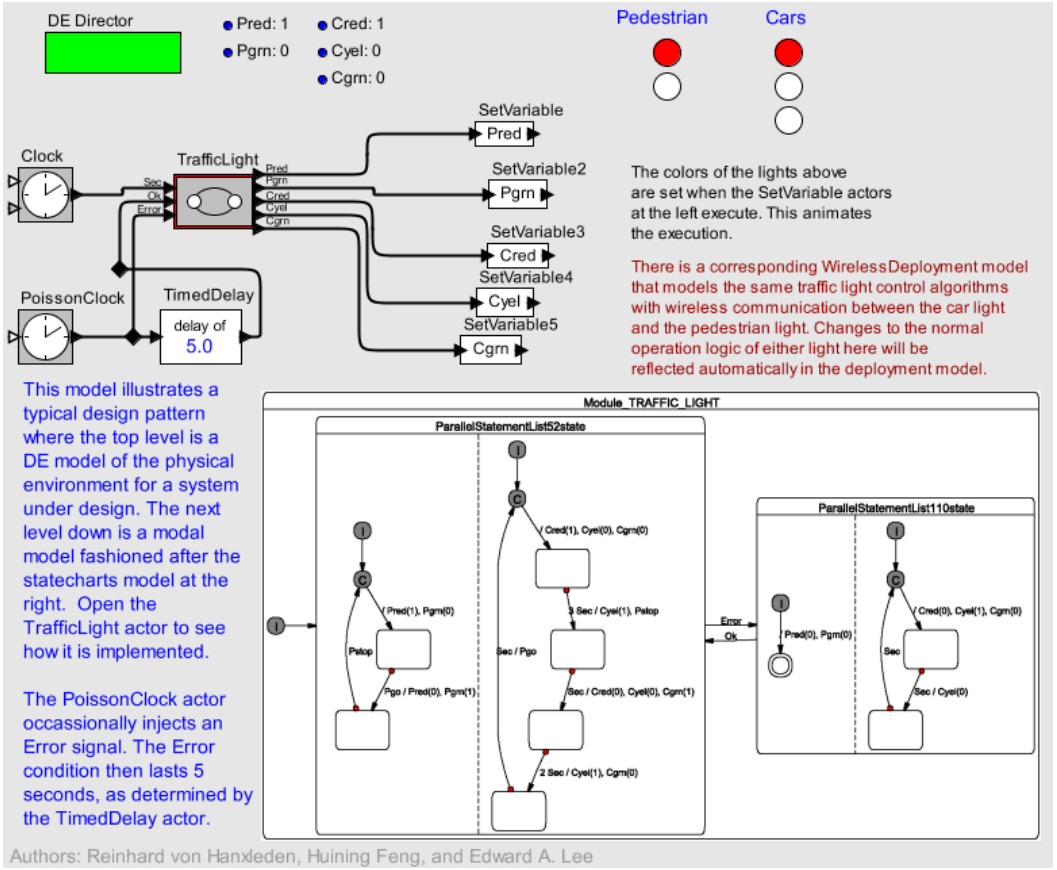
MoC: Synchronuous Dataflow

- 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



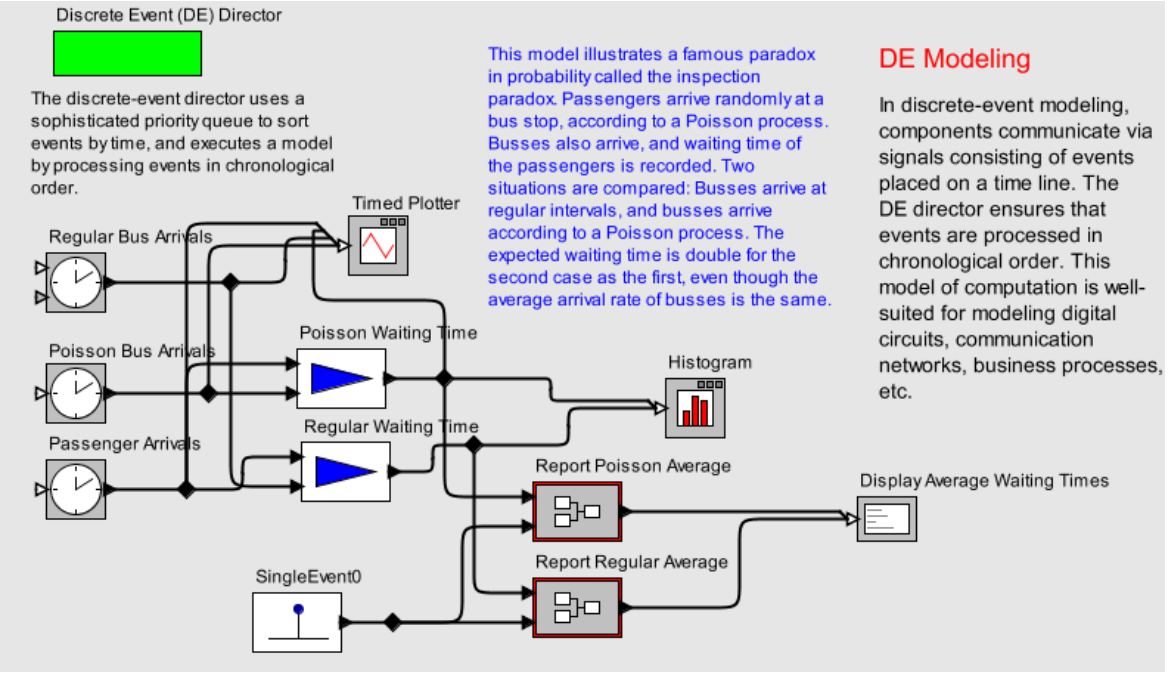
MoC: Synchronous Reactive

- Prinzip:
  - Annahme: unendlich schnelle Maschine
  - Ereignisse werden zyklisch verarbeitet (Ereignisse müssen nicht jede Runde eintreffen)
  - Pro Runde wird genau eine Reaktion berechnet
  - Häufig verwendet in Zusammenhang mit Finite State Machines
- Vorteile:
  - einfache formale Verifikation
- Werkzeuge:
  - Esterel Studio
  - Scade



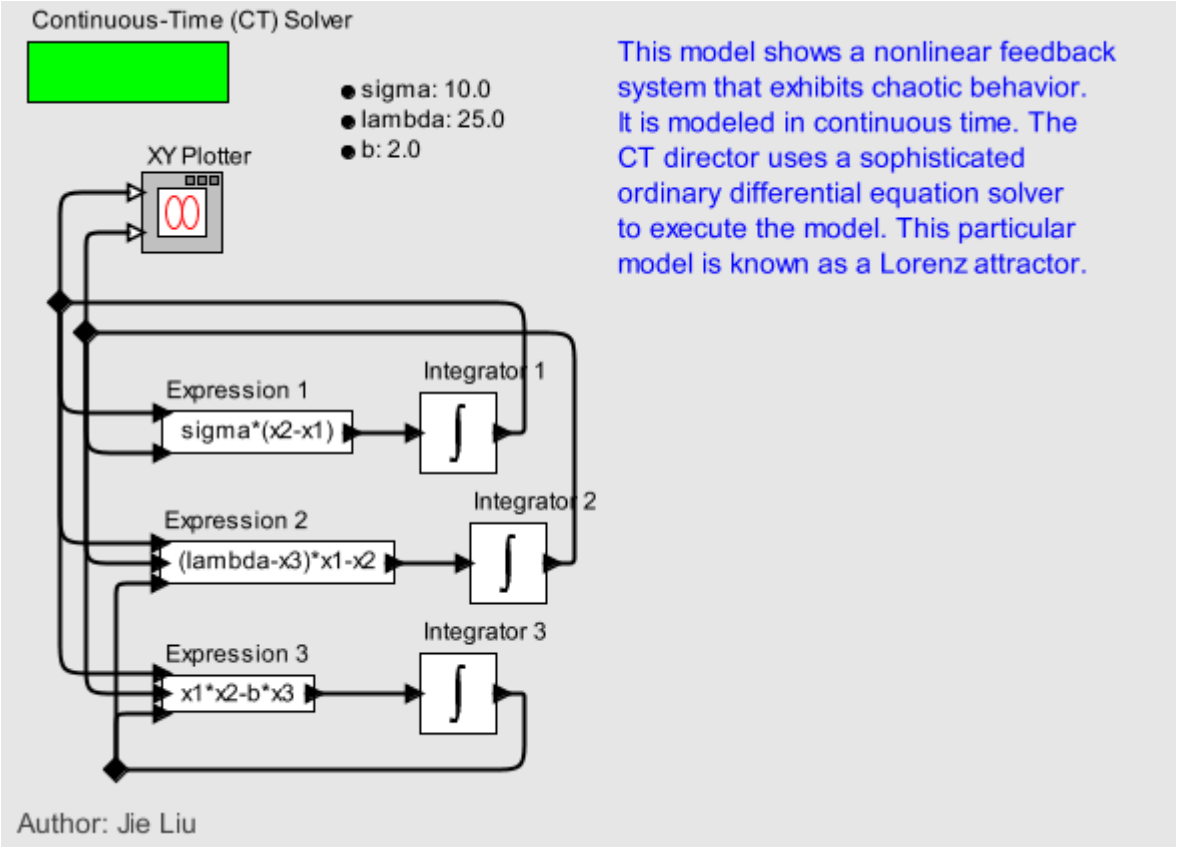
MoC: Discrete Event

- Prinzip:
  - Kommunikation über Ereignisse
  - Jedes Ereignis trägt einen Wert und einen Zeitstempel
- Anwendungsgebiet:
  - Digitale Hardware
  - Telekommunikation
- Werkzeuge:
  - VHDL
  - Verilog
- Varianten:
  - Distributed Discrete Events



MoC: Continuous Time

- Prinzip:
  - Verwendung kontinuierlicher Signale (zumeist Differential-gleichungen)
- Anwendungsgebiet:
  - Simulation
- Werkzeuge:
  - Simulink
  - Labview



Weitere MoCs

- Component Interaction:
  - Mischung von daten- und anfragegetriebener Ausführung
  - Beispiel: Web Server
- Discrete Time:
  - Erweiterung des synchronen Datenflussmodels um Zeit zwischen Ausführungen zur Unterstützung von Multiratensystemen
- Time-Triggered Execution
  - Die Ausführung wird zeitlich geplant
  - Anwendungsgebiet: kritische Regelungssysteme
  - Werkzeug: Giotto, FTOS
- Process Networks
  - Prozess senden zur Kommunikation Nachrichten über Kanäle
  - Kanäle können Nachrichten speichern

  $\Rightarrow$ asynchrone Nachrichten
  - Anwendungsgebiet: verteilte Systeme
- Rendezvous
  - synchrone Kommunikation verteilter Prozesse (Prozesse warten am Kommunikationspunkt bis Sender und Empfänger bereit sind)
  - Beispiele: CSP, CCS, Ada