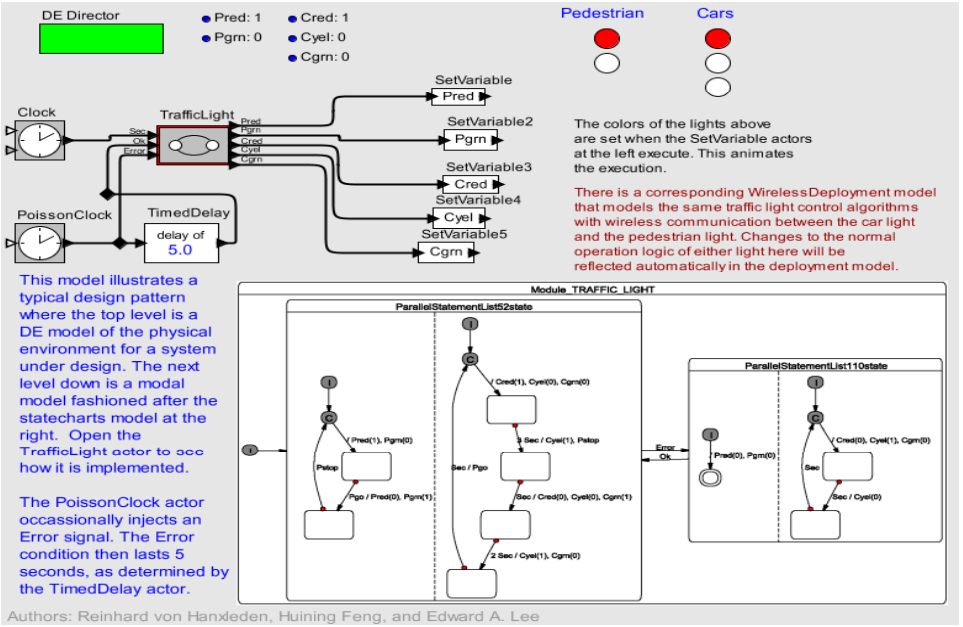
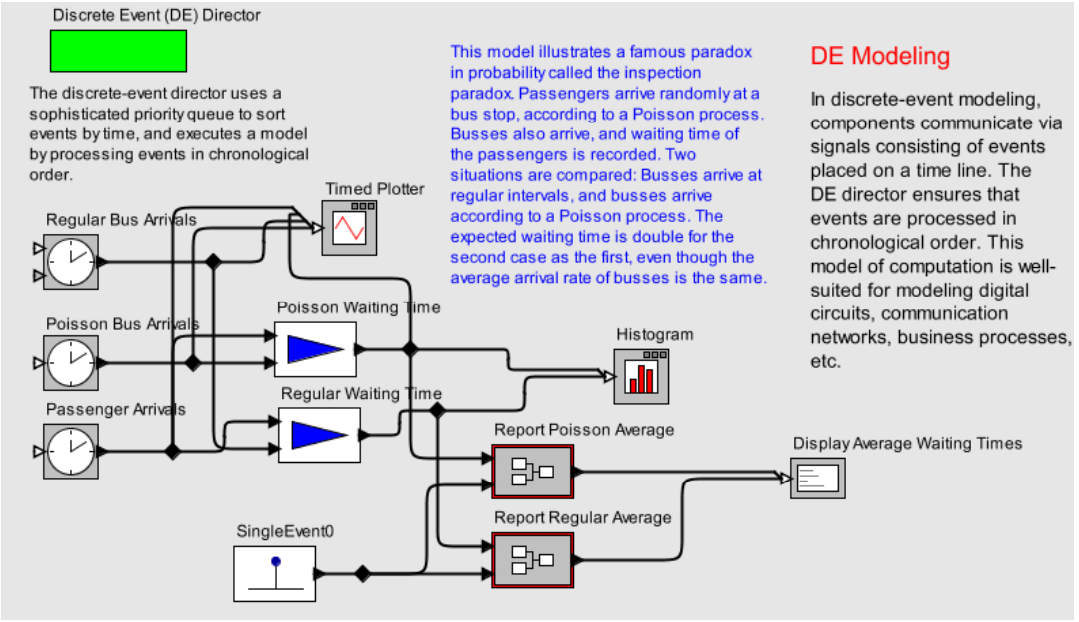
Example Ptolemy Model of Computation: Synchronous Reactive

- Prinzip:
  - Annahme: unendlich schnelle Maschine
  - Diskrete Ereignisse (DE) 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



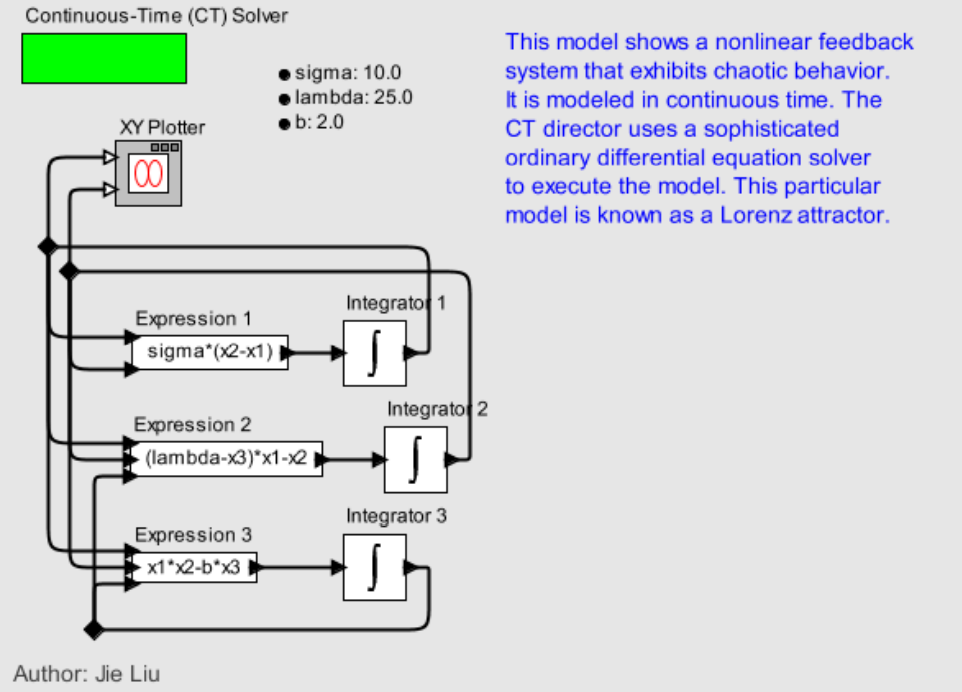
Example Ptolemy Model of Computation: Discrete Event

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



Example Ptolemy Model of Computation: Continuous Time

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



Weitere Models of Computation

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

Modellierung von Echtzeitsystemen

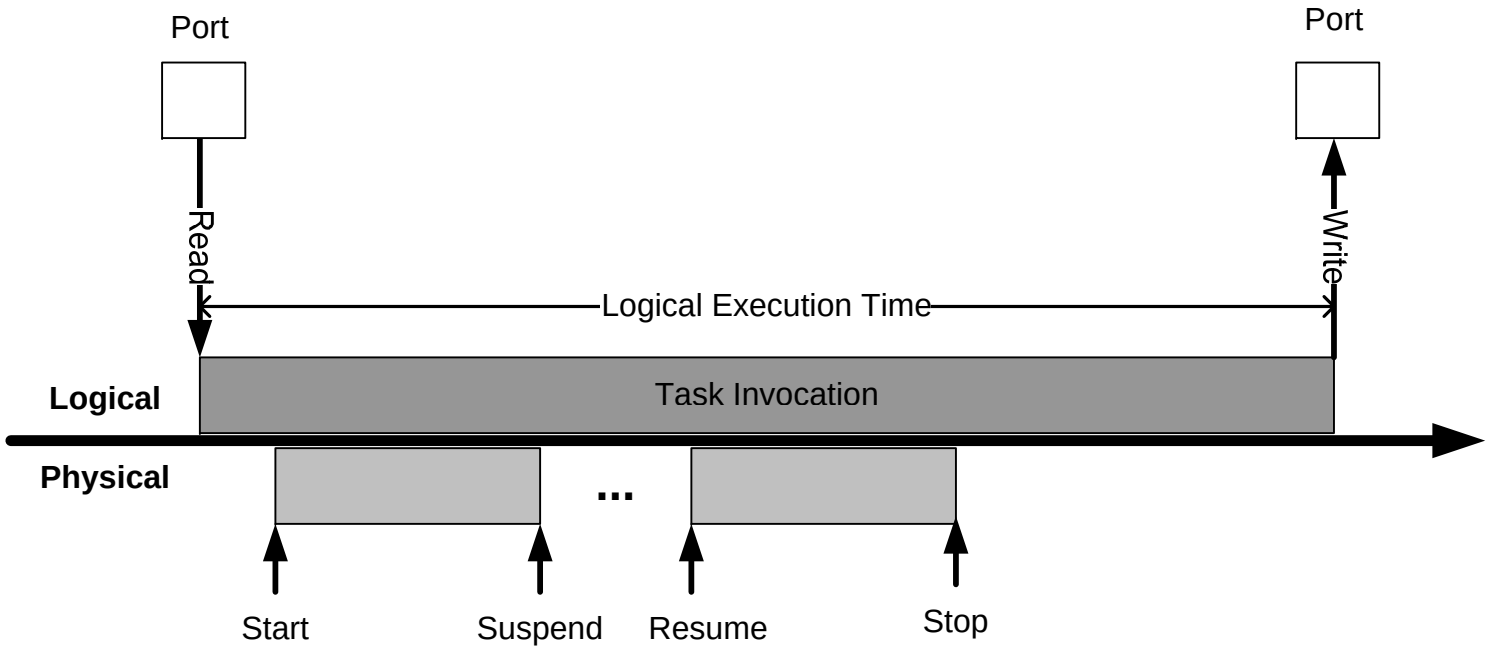
Zeitgesteuerte Systeme

Werkzeug: Giotto

Giotto: Hintergrund

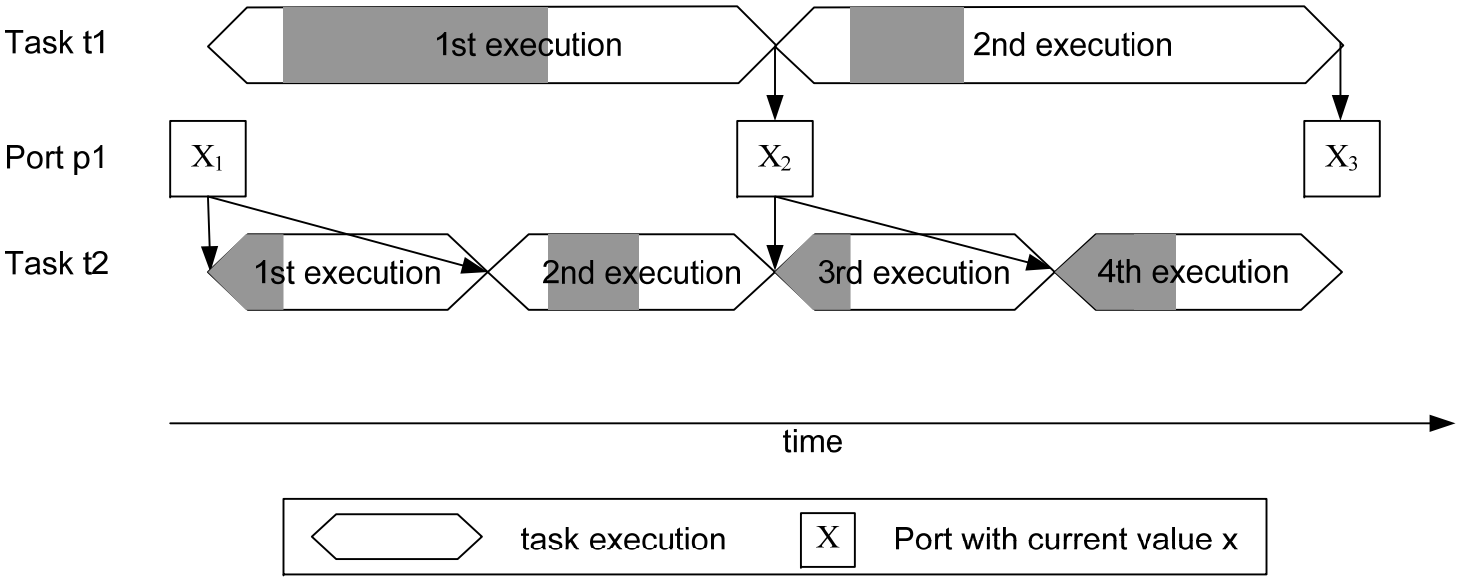
- Programmierumgebung für eingebettete Systeme (evtl. ausgeführt im verteilten System)
- Ziel:
  - strikte Trennung von plattformunabhängiger Funktionalität und plattformabhängigen Scheduling und Kommunikation
  - temporaler Determinismus
- Hauptkonzept: Logische Ausführungszeiten
- Aktoren:
  - Tasks
    - Programmblock aus sequentiellen Code
    - keine Synchronisationspunkte, blockende Operationen erlaubt
    - Schnittstellen: Ports
  - Drivers: realisieren die Kommunikation zwischen Ports
  - Flexibilität durch Modes/Guards
- Ausführung durch virtuelle Maschinen:
  - Embedded Machine: Reaktion der Tasks auf physikalische Ereignisse
  - Scheduling Machine: physikalisches Scheduling
- http://embedded.eecs.berkeley.edu/giotto/

Logische Ausführungszeit



Motivation siehe http://www.cs.uic.edu/~shatz/SEES/henzinger.slides.ppt

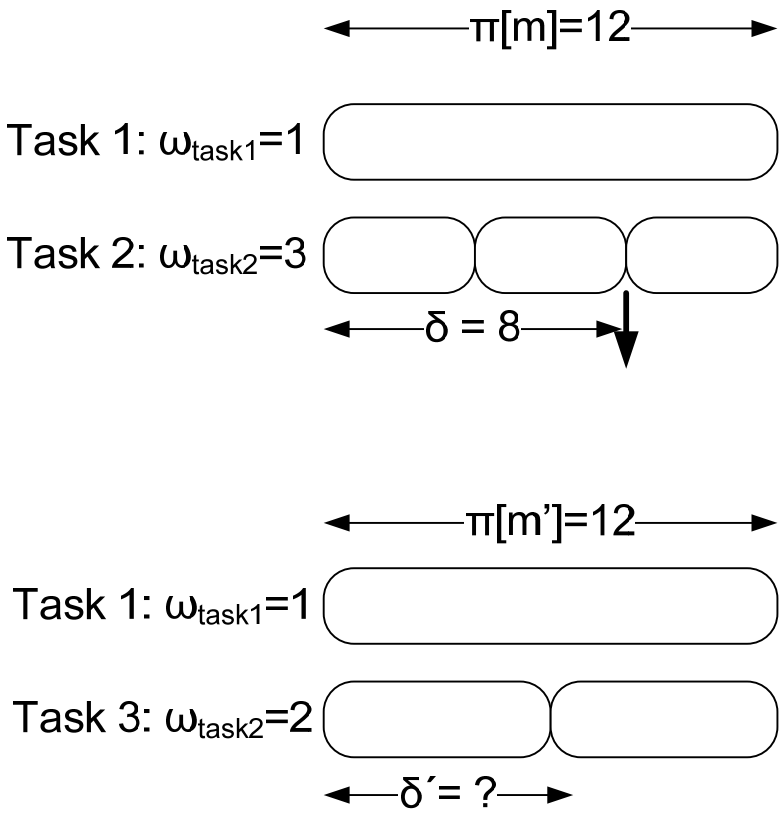
Kommunikation zwischen Tasks



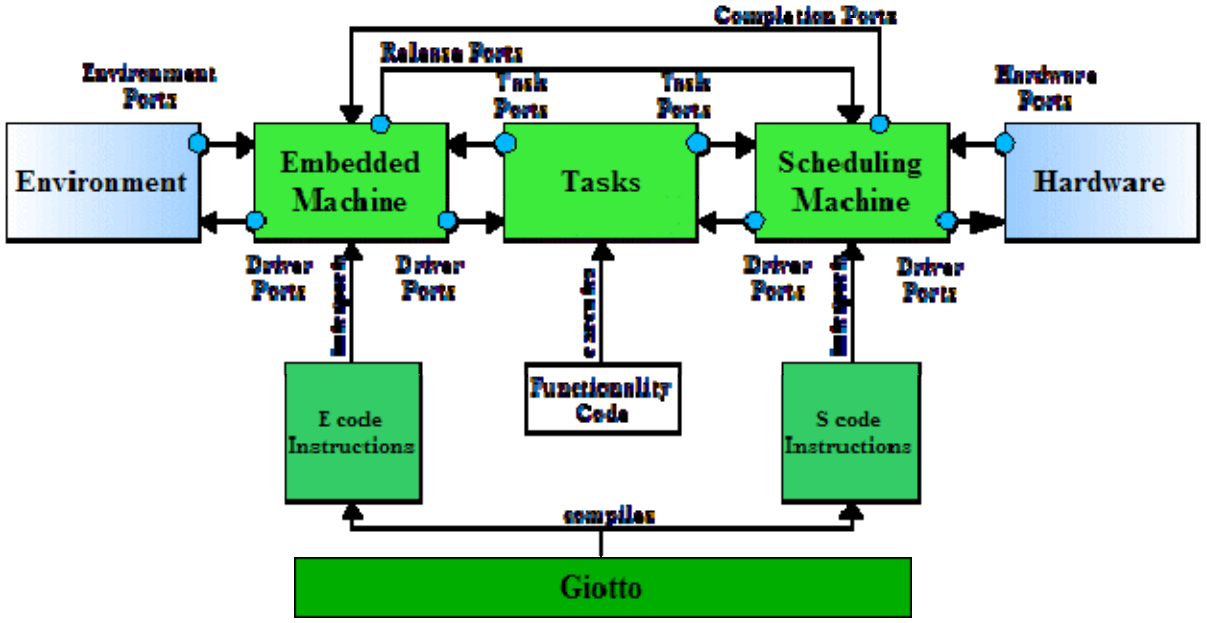
Modes / Guards

- Um die Ausführung flexibel zu gestalten, bietet Giotto Modes und Guards an
  - Guards: Boolsche Funktion, die über die Ausführung eines Tasks entscheidet (wird vor Start des Tasks aufgerufen)
  - Mode: Menge von Tasks und Drivers die zeitgleich ausgeführ werden, es kann immer nur ein Mode aktiv sein.
- Nicht-harmonischer Moduswechsel (Unterbrechung eines laufenden Modes):
  - Voraussetzung: $\pi[m]/\omega_{task} = \pi[m']/\omega'_{task}$
    m: Quellmodus, m': Zielmodus, $\pi[m]$: Modusdauer m, ω_{task}: Taskfrequenz $\Rightarrow$ Logische Ausführungszeit muss gleich sein
  - Wechselmechanismus:

    γ = LCM $\{\pi[m]/\omega_{task} | \{\omega_{task}, t,) \in$ Invokes[m],
    $\delta' = \pi[m']-(\varepsilon-\delta)$ mit $\varepsilon = n^* \gamma \geq \delta$
    LCM: least common multiple, δ: aktuelle Rundenzeit, δ': neue Rundenzeit in m', $\varepsilon - \delta$: Zeit bis zum nächsten gleichzeitigen Beendigungspunkt



Ausführungsumgebung



Zusammenfassung

- Das Konzept der logischen Ausführungszeiten erlaubt eine Abstrahierung von der physikalischen Ausführungszeit und somit die Trennung von plattformunabhängigem Verhalten (Funktionalität und zeitl. Verhalten) und plattformabhängiger Realisierung (Scheduling, Kommunikation)
- Die Ausführung erfolgt über zwei virtuelle Maschinen:
  - E-Machine: Interaktion mit der Umgebung (reaktiv)
  - S-Machine: Interaktion mit der ausführenden Plattform (proaktiv), Vorteil: Schedule kann vorab berechnet werden
- Weitere Literaturhinweise:
  - Henzinger et al.: Giotto: A time-triggered language für embedded programming, Proceedings of the IEEE, vol.91, no.1, pp. 84-99, Jan 2003
  - Henzinger et al.: Schedule-Carrying Code, Proceedings of the Third International Conference on Embedded Software (EMSOFT), 2003