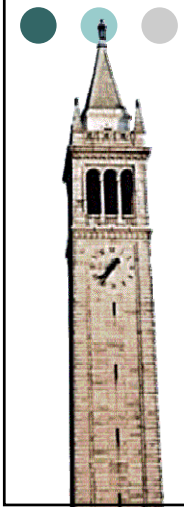


Parallel, Concurrent, and Distributed Software in Cyber-Physical Systems

Edward A. Lee
Robert S. Pepper Distinguished Professor
UC Berkeley

Invited Talk
Int. Workshop on User-Centric Cyber-Physical Systems and Services (UC-CPS)

Institute of Information Science, Academia Sinica
Taipei, Taiwan, December 8-9, 2009

Motivation

Much effort in computer science has gone into attempting to make computers behave like humans.

My focus is on making computers behave like physical processes so that humans that interact with them intuitively, in the same manner with which they interact with their physical environment.

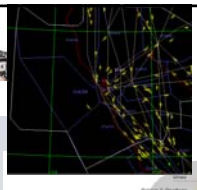
Abstract

Parallel, concurrent, and distributed software plays a key role in user-centric cyber-physical systems. It handles a multiplicity of streams of sensor data, extracts and fuses models of the physical environment, and coordinates distributed reactions. Humans require that such software behave in ways that would be expected of physical processes. Achieving that illusion, however, is challenging using today's prevailing technologies for software design. These technologies are rooted in abstractions that have only poor analogies in the physical world. This talk will critically examine these abstractions and suggest replacements. The goal is software design techniques that naturally lead to software behaviors that emulate physical processes.

Cyber-Physical Systems (CPS):

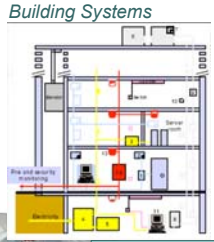
Orchestrating networked computational resources with physical systems

Avionics

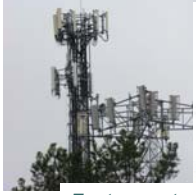


Transportation (Air traffic control at SFO)

Building Systems

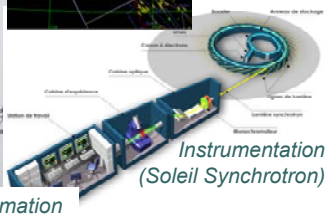


Telecommunications



Automotive

E-Corner, Siemens



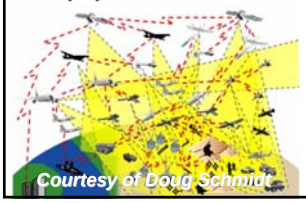
Instrumentation (Soleil Synchrotron)

Daimler-Chrysler

Power generation and distribution

Courtesy of General Electric

Military systems:



Courtesy of Doug Schmidt

Factory automation

Courtesy of Kuka Robotics Corp.

CPS Example – Printing Press

Bosch-Rexroth

- *High-speed, high precision*
  - *Speed: 1 inch/ms*
  - *Precision: 0.01 inch*
    -> Time accuracy: 10us
- *Open standards (Ethernet)*
  - *Synchronous, Time-Triggered*
  - *IEEE 1588 time-sync protocol*
- *Application aspects*
  - *local (control)*
  - *distributed (coordination)*
  - *global (modes)*

Even without distributed computing, timing can get complex.

Consider an automotive engine controller.

Embedded software using timers, interrupts, threads, shared memory, priorities, and mutual exclusion can realize such systems. But how hard is it to get right?

Standard approaches to concurrency and real time rely on threads, priorities, mutexes, etc...

Sutter and Larus observe:

> *"humans are quickly overwhelmed by concurrency and find it much more difficult to reason about concurrent than sequential code. Even careful people miss possible interleavings among even simple collections of partially ordered operations."*

H. Sutter and J. Larus. Software and the concurrency revolution. ACM Queue, 3(7), 2005.

Is Concurrency Hard?

It is not concurrency that is hard…

…It is Threads that are Hard!

Threads are sequential processes that share memory. From the perspective of any thread, the entire state of the universe can change between any two atomic actions (itself an ill-defined concept).

Imagine if the physical world did that…

Concurrent programs using shared memory are incomprehensible because concurrency in the physical world does not work that way.

We have no experience!

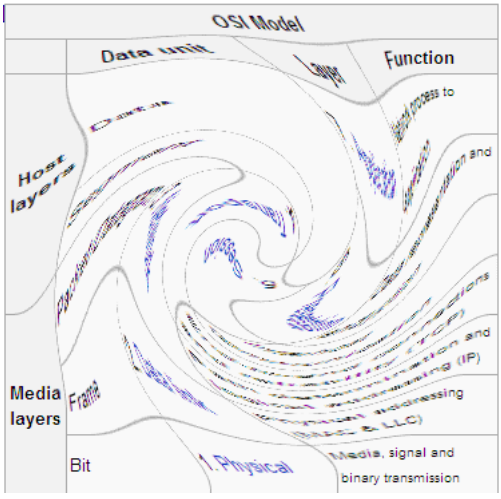
For distributed applications, the problem gets harder. Networks with "quality of service" are insufficient. Need "correctness of service."

Traditionally, "***faster* is *better***."

This is like saying that for a roller coaster, "**stronger is better**."

We have to change the mindset to "**not fast enough is *an error***!"

Abstraction Layers in Networks



The point of these abstraction layers is to isolate a system designer from the details of the implementation below, and to provide an abstraction for other system designers to build on.

In today's general-purpose networks, timing is a property that emerges from the details of the implementation, and is not included in the abstractions. *For time-critical applications, the abstraction layers fail.*

For distributed cyber-physical systems,

Timing needs to be a part of the network *semantics*, not a side effect of the implementation.

Technologies needed:

- Time synchronization
- Bounds on latency
- Time-aware fault isolation and recovery
- Time-aware robustness

Background - Domain-Specific Networks with Timed Semantics

- **WorldFIP** (Factory Instrumentation Protocol)
  - Created in France, 1980s, used in train systems
- **CAN**: Controller Area Network
  - Created by Bosch, 1980s/90s, ISO standard
- Various **ethernet** variants
  - PROFInet, EtherCAT, Powerlink, …
- **TTP/C**: Time-Triggered Protocol
  - Created around 1990, Univ. of Vienna, supported by TTTech
- **MOST**: Media Oriented Systems Transport
  - Created by a consortium of automotive & electronics companies
  - Under active development today
- **FlexRay**: Time triggered bus for automotive applications
  - Created by a consortium of automotive & electronics companies
  - Under active development today

Services Provided by Networks with Timed Semantics

- Frequency locking
- Time synchronization
- Bounded latency
- Fault isolation (sometimes)
- Priorities (sometimes)
- Admission control (sometimes)

Not so Domain-Specific Network Mechanisms

- Frequency locking
  - E.g., **synchronous ethernet**: ITU-T G.8261, May 2006
  - Enables integrating circuit-switched services on packet-switched networks
  - Can deliver performance independent of network loading.

Press Release

Zarlink Semiconductor Corp.

Release date: January 31, 2007

Zarlink and Marvell® First to Demonstrate Synchronous Ethernet Solution Supporting Network-Quality Performance

Companies demonstrate synchronization over Ethernet physical layer using Zarlink PLL (phase locked-loop) and Marvell Ethernet PHY technologies

OTTAWA, Jan. 31 /- Zarlink Semiconductor (NYSE/TSX:ZL) and Marvell® (NASDAQ:MRVL) today announced the successful demonstration of a synchronous Ethernet solution using already available products from both companies that will allow carriers to support real-time services over packet-based networks.

- Time synchronization
  - E.g., **IEEE 1588** standard set in 2002.
  - Synchronized time-of-day across a network.

Time Synchronization on Ethernet with TCP/IP: IEEE 1588 PTP

Press Release October 1, 2007

National Semiconductor
The Sight & Sound of Information

NEWS RELEASE

For More Information Contact

Media Contact
Naomi Mitchell
National Semiconductor
(408) 721-2142
naomi.mitchell@nsc.com

Reader Information
Design Support Group
(800) 272-9959
www.national.com

Industry's First Ethernet Transceiver with IEEE 1588 PTP Hardware Support from National Semiconductor Delivers Outstanding Clock Accuracy

Using DP83640, Designers May Choose Any Microcontroller, FPGA or ASIC to Achieve 8- Nanosecond Precision with Maximum System Flexibility

Clocks on a LAN agree on the current time of day to within 8ns, far more precise than older techniques like NTP.

A question we are addressing at Berkeley: How does this change how we develop distributed real-time software?

A Programming Model for Distributed Cyber-Physical Systems

The question we address:

Given a common notion of time shared to some known precision across a network, and given bounded network latencies, can we design better distributed embedded software?

Our answer (today):

Use discrete-event (DE) models for specification of systems, bind *model time* to *real time* only exactly where this is needed.

My Agenda

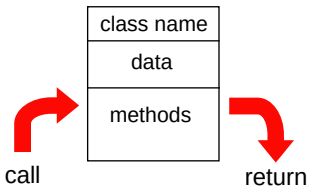
I will show a particular approach to the design of concurrent and distributed time-sensitive systems that is an *actor-oriented* component technology, with a timed concurrency model that has good physical intuition, and that can be used to define distributed real-time systems.

The approach is called PTIDES (pronounced “tides”), for Programming Temporally Integrated Distributed Embedded Systems.

See: Zhao, Lee and Liu "A Programming Model for Time-Synchronized Distributed Real-Time Systems," RTAS 2007.

Object Oriented vs. Actor Oriented Software Component Technologies

The established: Object-oriented:

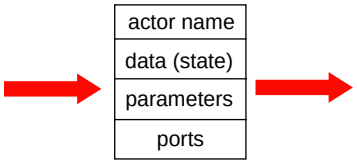


What flows through an object is sequential control

Things happen to objects

The alternative: Actor oriented:

Actors make things happen



What flows through an object is evolving data

Some Actor-Oriented Influences

- BIP [Basu, Bozga, Sifakis 2006]
- Colif [Jerraya et al. 2001]
- Esterel [Berry et al. 1992]
- ForSyDe [Sander, Jantsch 2004]
- FunState [Thiele, Ernst, Teich, et al. 2001]
- Giotto [Henzinger et al. 2001]
- HetSC [Herrera, Villar 2006]
- LabVIEW [Kodosky et al. 1986]
- Lustre [Halbwachs, Caspi et al. 1991]
- Metropolis [Goessler, Sangionvanni-Vincentelli et al. 2002]
- Model Integrated Computing [Sztipanovits, Karsai, et al. 1997]
- Ptolemy Classic [Buck, Ha, Messerschmitt, Lee et al. 1994]
- Ptolemy II [Eker, Janneck, Lee, et al. 2003]
- RTComposer [Alur, Weiss 2008]
- SCADE [Berry et al. 2003]
- SDL [Various, 1990s]
- Signal [Benveniste, Le Guernic 1990]
- Simulink [Ciolfi et al., 1990s]
- Statecharts [Harel 1987]

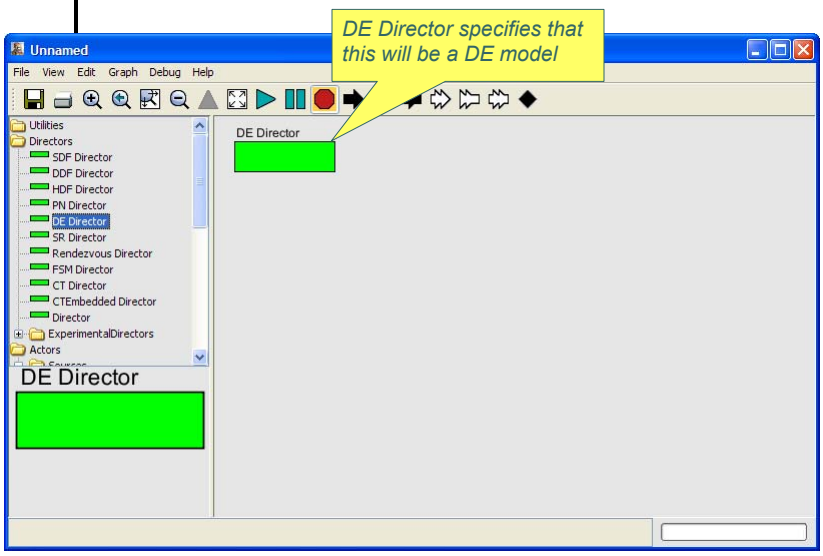
Our Approach is based on Discrete Events (DE)

- Concurrent actors
- Exchange time-stamped messages ("events")

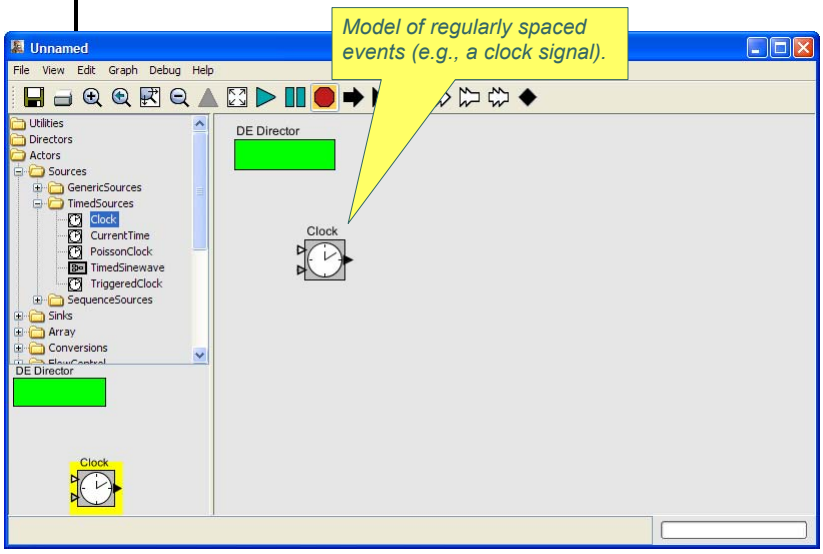
A correct execution is one where every actor reacts to input events in time-stamp order.

Time stamps are in "model time," which typically bears no relationship to "real time" (wall-clock time). We use *superdense time* for the time stamps.

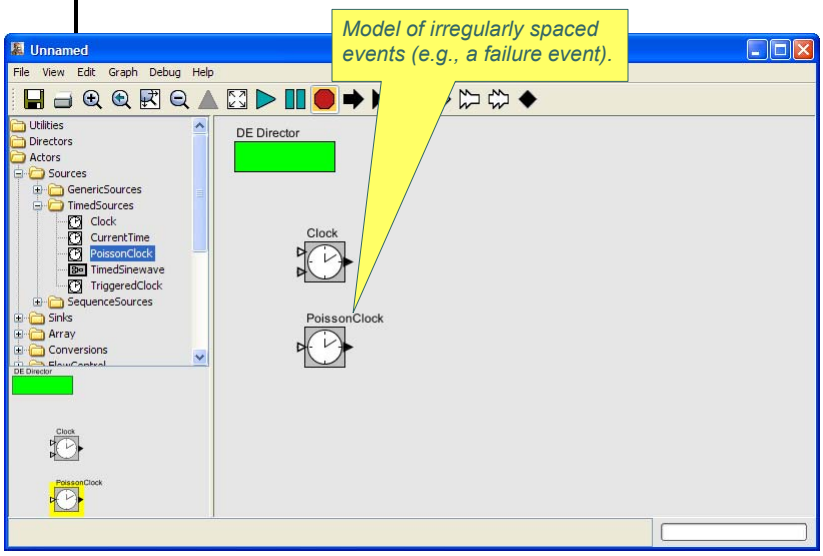
Example DE Model (in Ptolemy II)



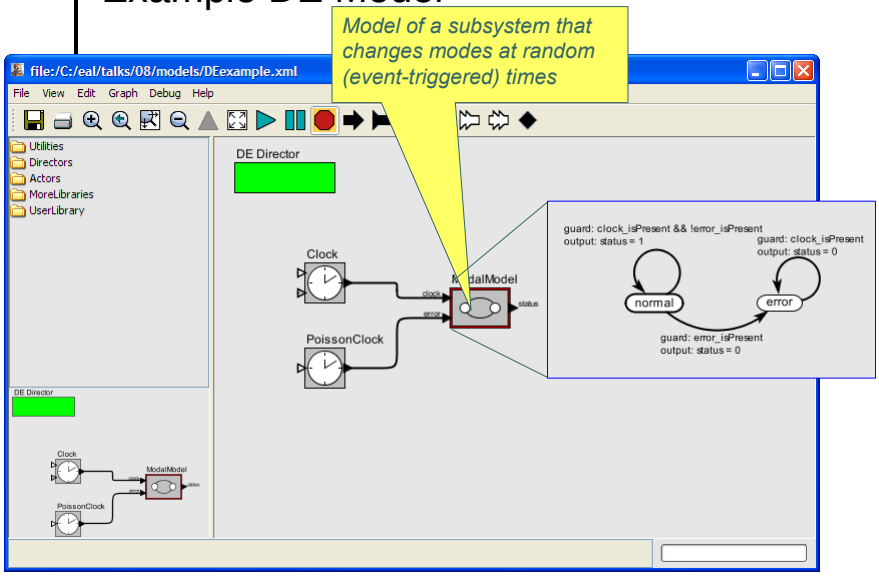
Example DE Model



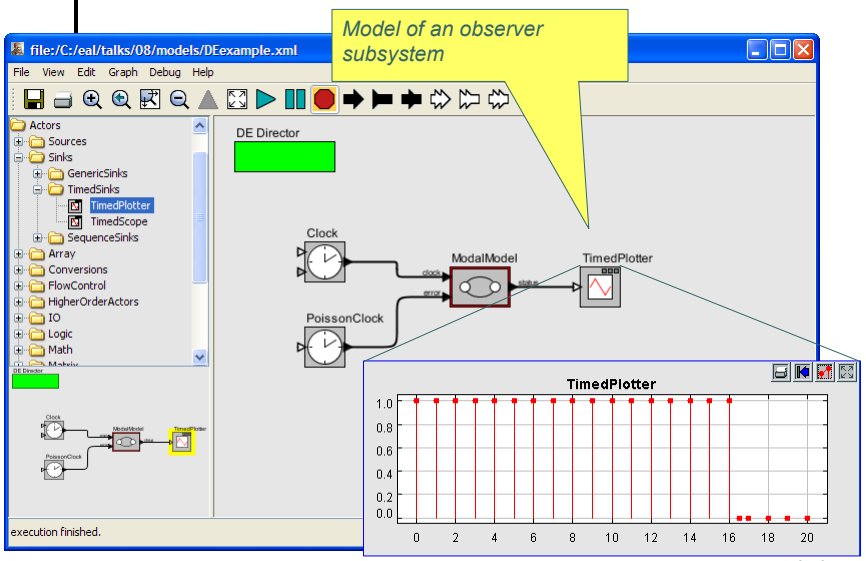
Example DE Model



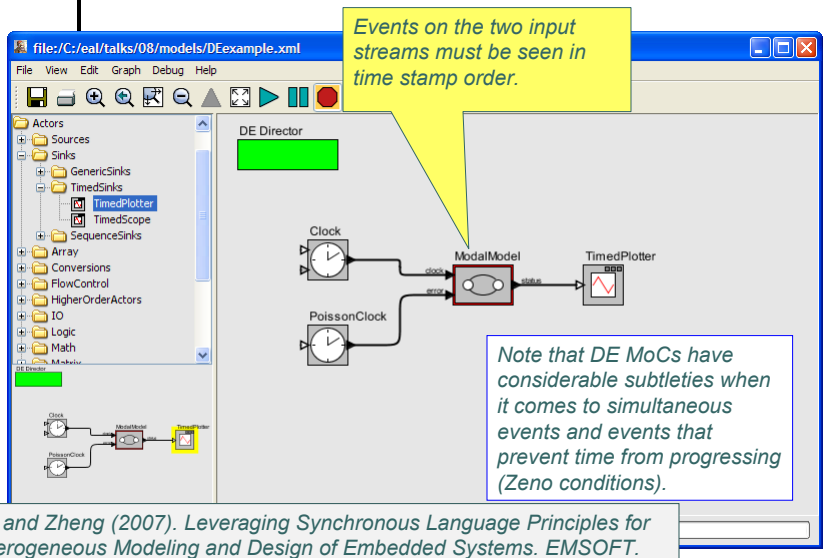
Example DE Model



Example DE Model

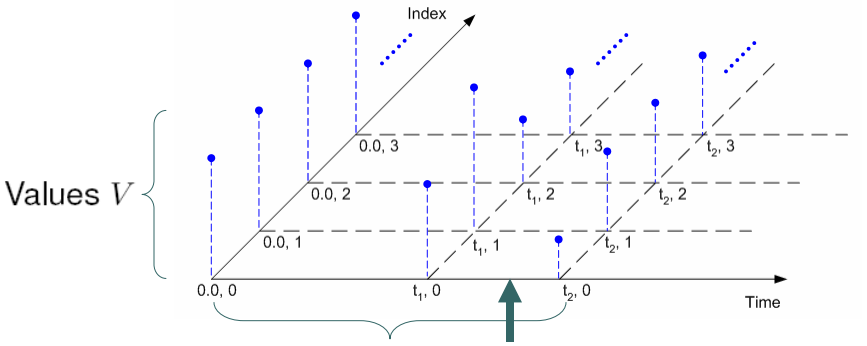


Example DE Model



Lee and Zheng (2007). Leveraging Synchronous Language Principles for Heterogeneous Modeling and Design of Embedded Systems. EMSOFT.

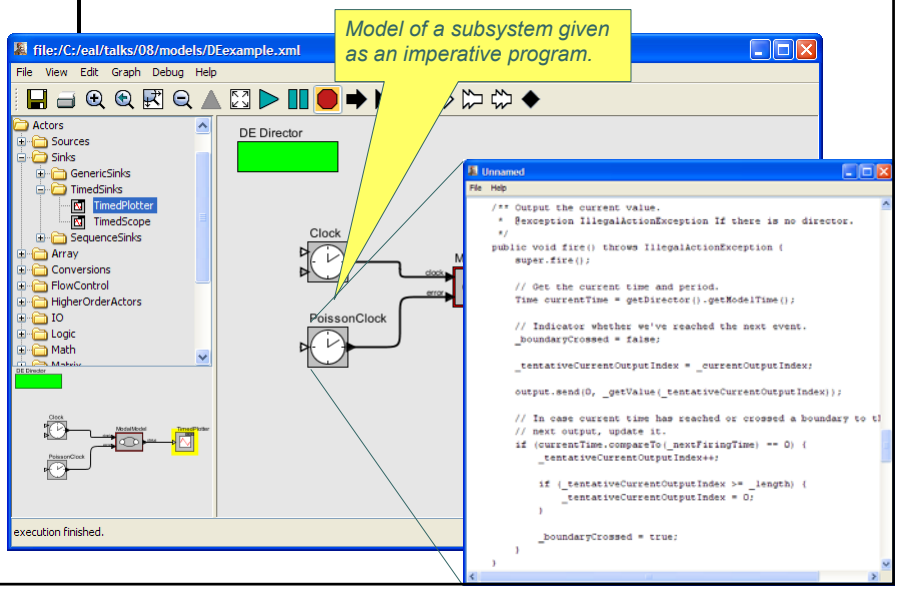
Aside:
Superdense Time Enables Better Conjunction of Computation and Physical Processes



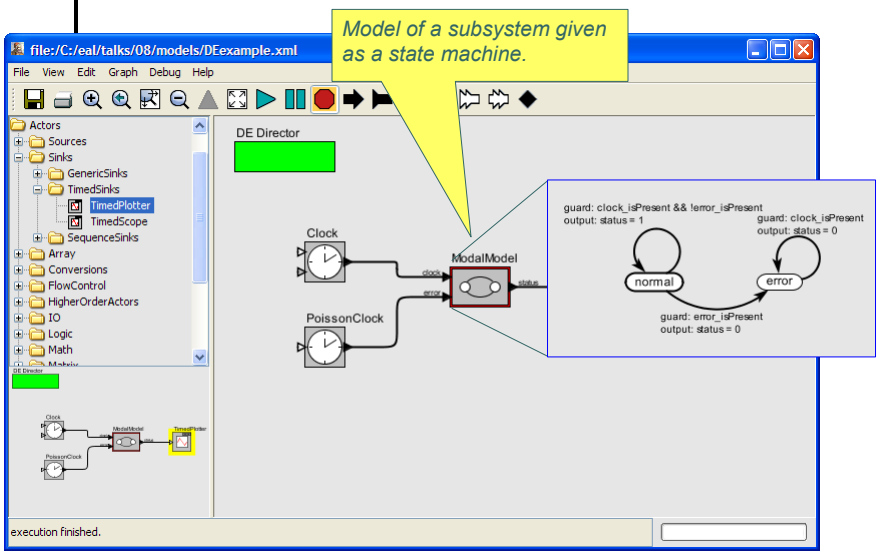
Initial segment $I \subseteq \mathbb{R}_+ \times \mathbb{N}$ where the signal is defined

Absent: $s(\tau) = \varepsilon$ for almost all $\tau \in I$.

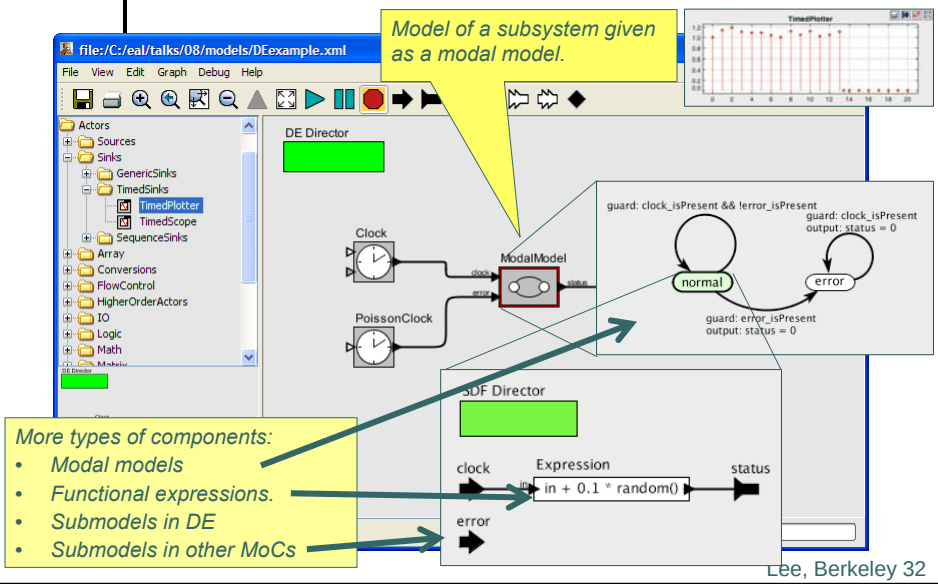
This is a Component Technology



This is a Component Technology



This is a Component Technology



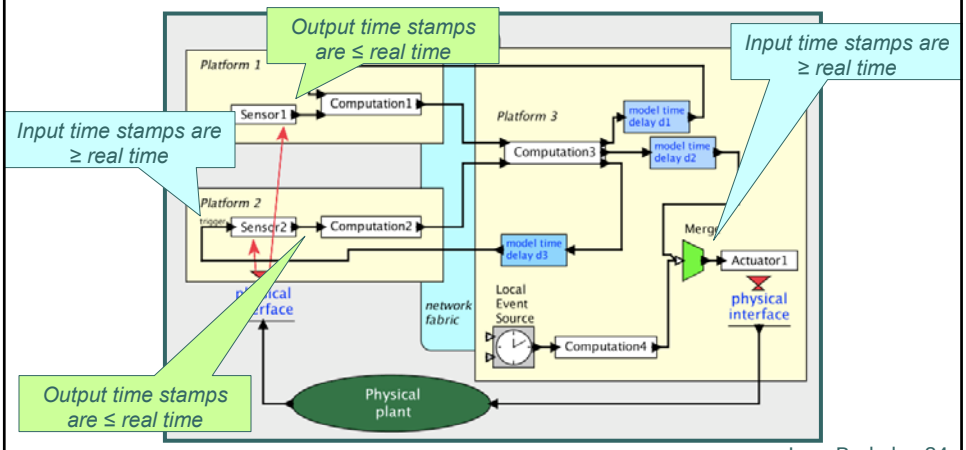
Using DE Semantics in Distributed Real-Time Systems

- DE is usually a simulation technology.
- Distributing DE is done for acceleration.
- Hardware design languages (e.g. VHDL) use DE where time stamps are literally interpreted as real time, or abstractly as ticks of a physical clock.

- We are using DE for distributed real-time software, binding time stamps to real time only where necessary.
- *PTIDES*: Programming Temporally Integrated Distributed Embedded Systems

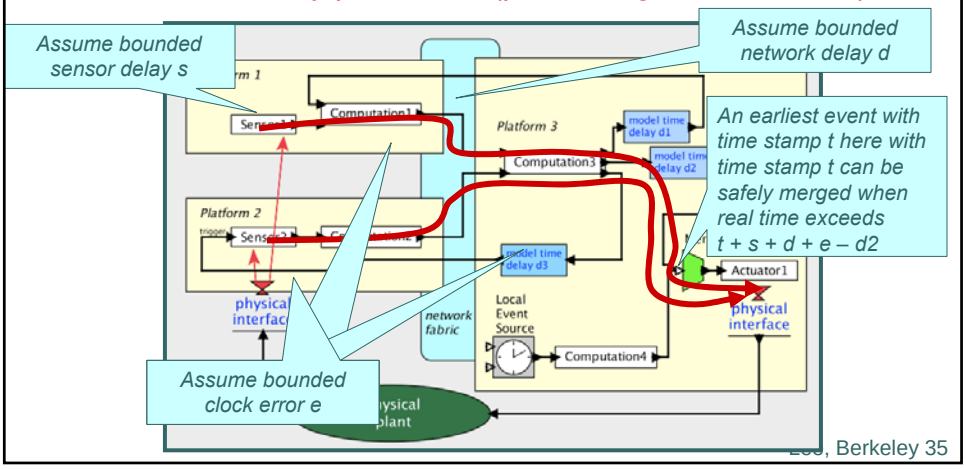
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Distributed execution under discrete-event semantics, with "model time" and "real time" bound at sensors and actuators.



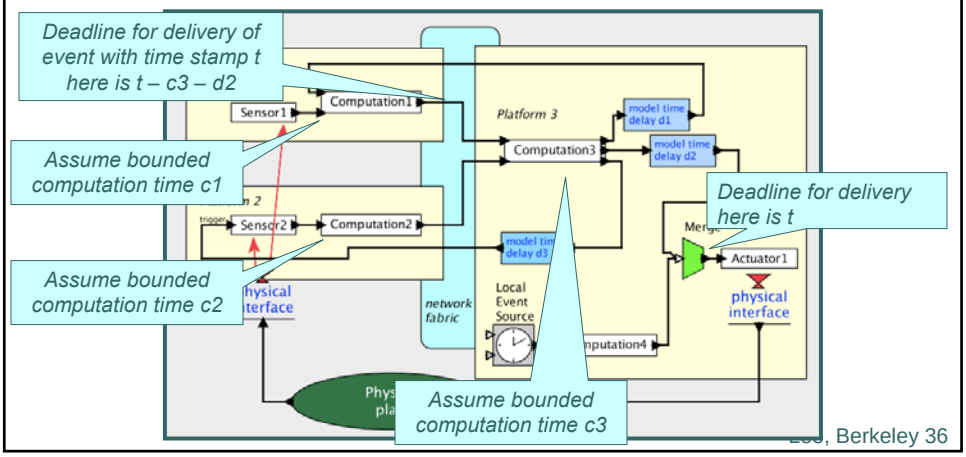
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

PTIDES uses static causality analysis to determine when events can be safely processed (preserving DE semantics).



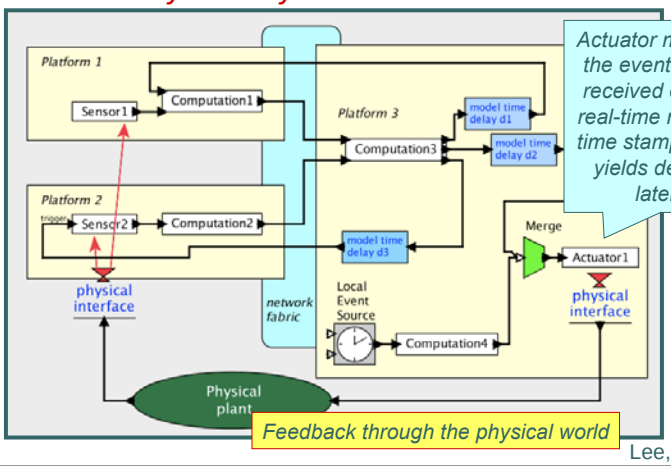
PTIDES: Programming Temporally Integrated Distributed Embedded Systems

Schedulability analysis incorporates computation times to determine whether we can guarantee that deadlines are met.

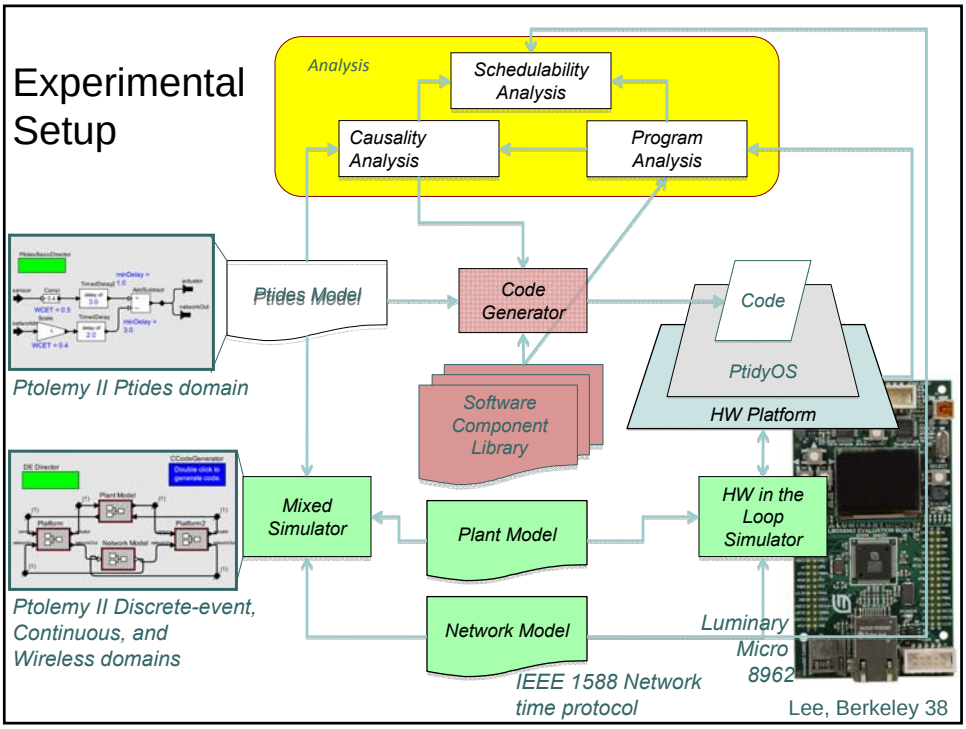


PTIDES: Programming Temporally Integrated Distributed Embedded Systems

… and being explicit about time delays means that we can analyze control system dynamics…



Experimental Setup



Summary

- Cyber-physical systems create new research opportunities.
- The concurrency problem requires breaking away from threads.
- The networking problem requires timing to be a correctness property rather than a quality of service consideration.
- The PTIDES model of computation offers an attractive possible programming model for distributed cyber-physical systems.

The Ptolemy Pteam