Mixing Models of Computation

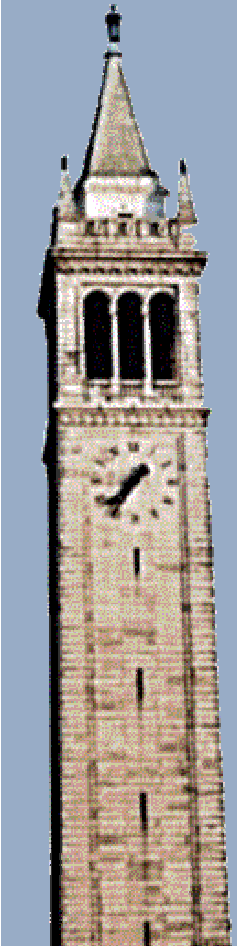
Jie Liu

Palo Alto Research Center (PARC)

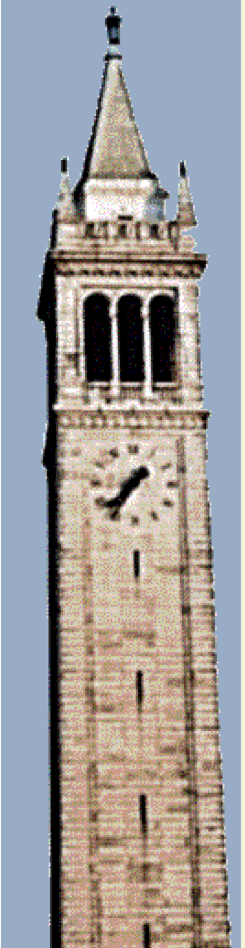
3333 Coyote Hill Rd., Palo Alto, CA 94304

liuj@parc.com

joint work with Prof. Edward A. Lee and the Ptolemy team in the MoBIES project



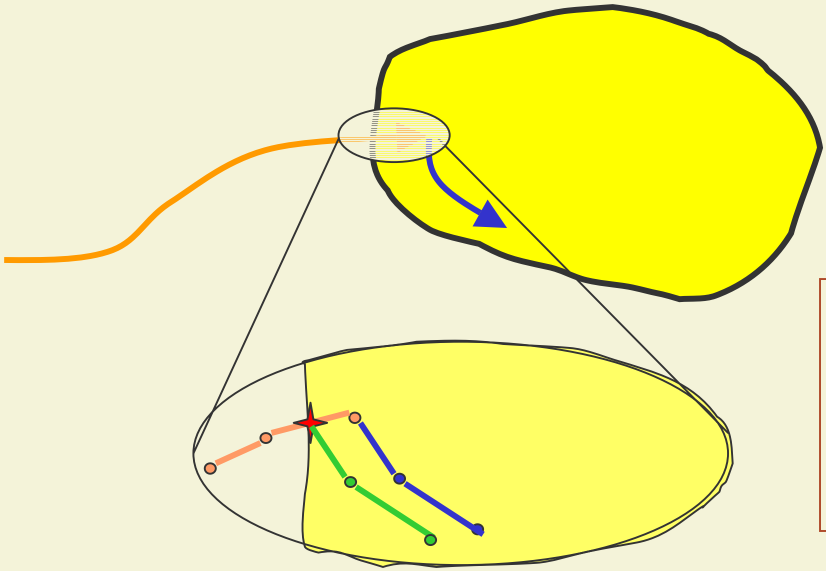
Mixing Models of Computation



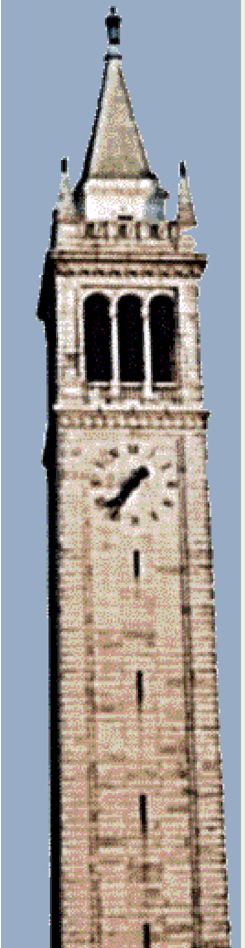
- Tool integration is about semantics integration
  - Tools essentially reflect the models of computation they implement or assume.
    - Simulink – continuous-time/mixed signal
    - Charon – hybrid automata
    - Teja – timed automata
    - Giotto – time triggered architecture
    - ns (network simulator) – discrete event
    - Esterel – synchronous/reactive
    - …
  - Not all semantic models are interchangeable
  - Not all semantic models are compositional
  - Not all tools are developed to work with other tools
- Ptolemy II is a framework to study semantics integration

Example #1: Precise Event Detection

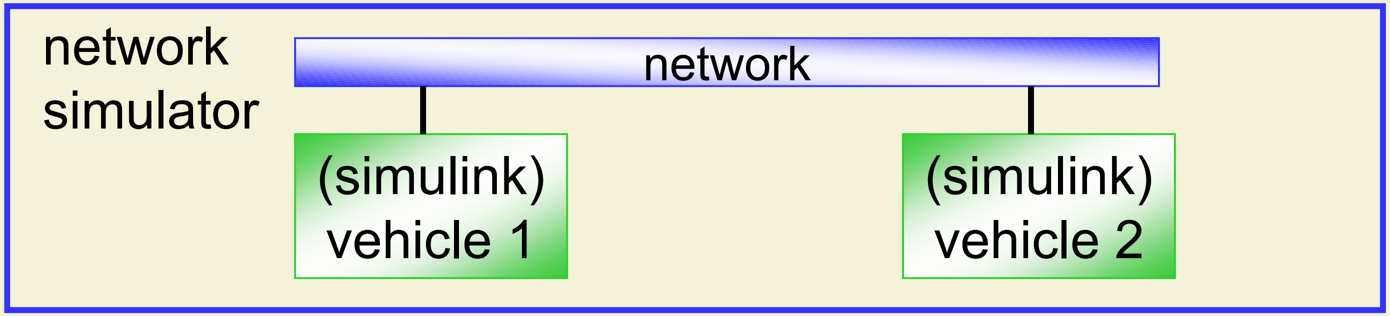
- In a mixed-signal/hybrid system model, not all discrete events are predictable.
- Events that depend on the value of continuous state variables (like zero crossing) need to be iteratively detected through numerical integration.



Simulink, Charon, and Ptolemy II support precise event detection; while Teja does not support it for good reason.



Example #2: Causality and Rollback



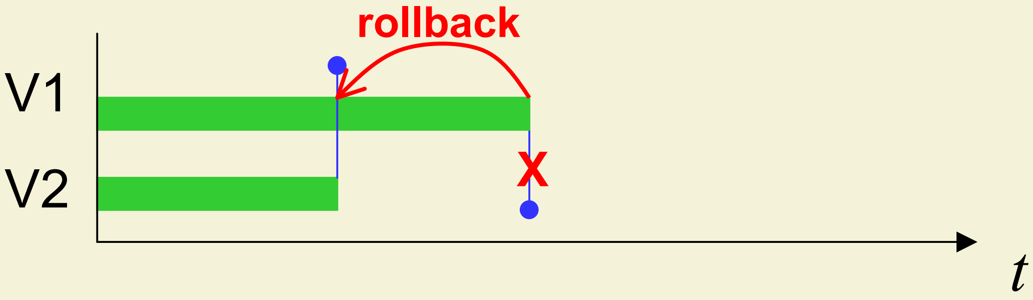
network simulator

network

(simulink) vehicle 1

(simulink) vehicle 2

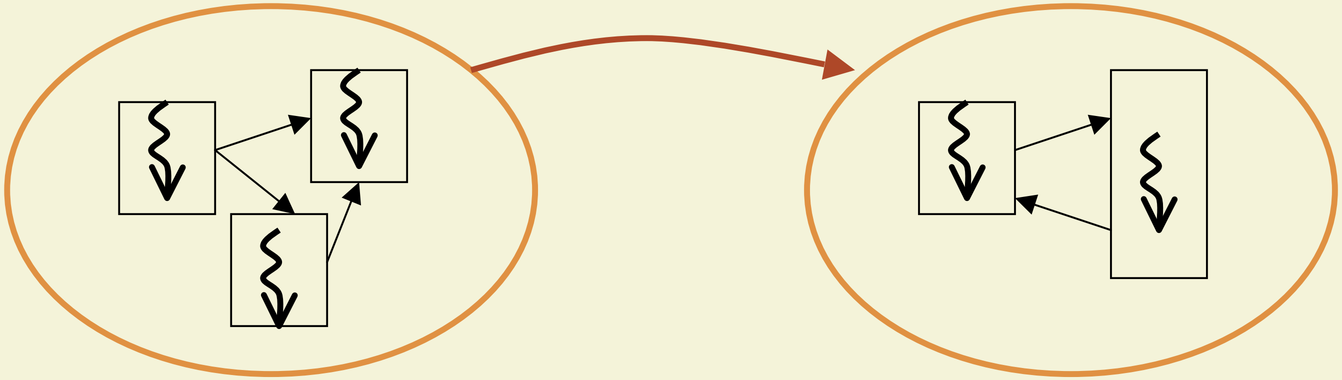
- How to manage the progression of time in three tools?



- Most continuous-time/mixed-signal tools do not support rollback

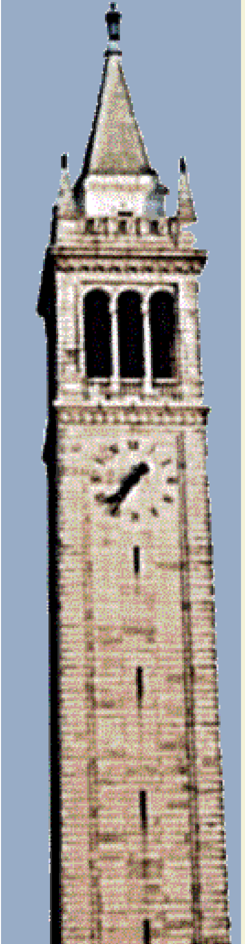
Example #3: Precise Mode Switching

(A scenario learned from the SEC project)



- When perform a mode switching or a reconfiguration, how to pause/turn off existing threads safely?
- Not all executions return their flow of control
- Not all executions return their flow of control at quiescent states.

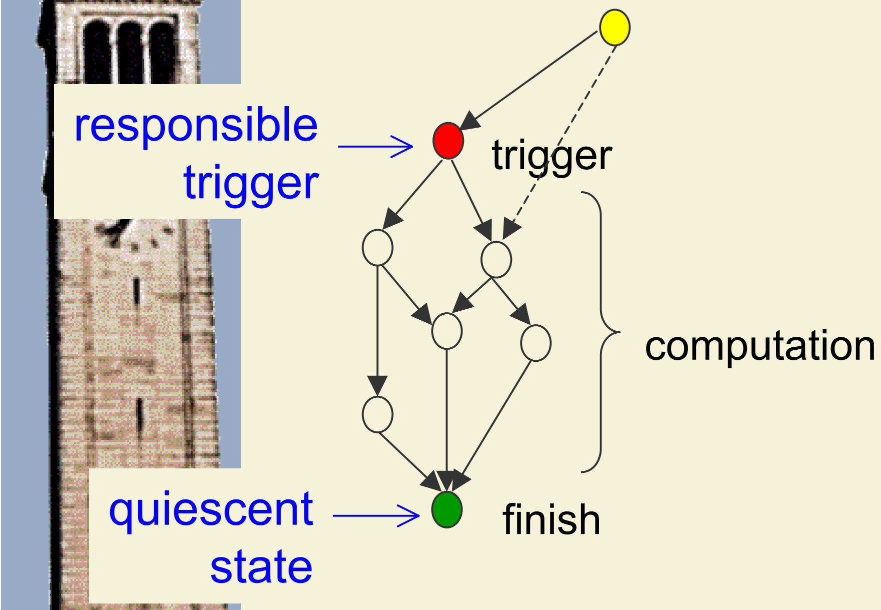
The Ptolemy Approach

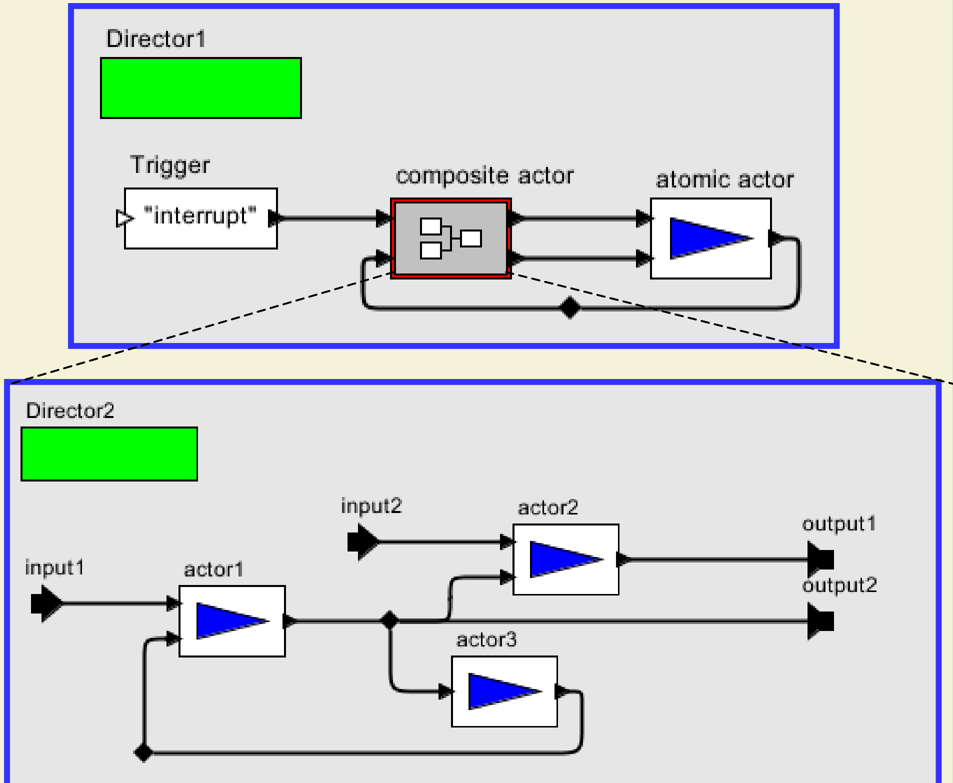


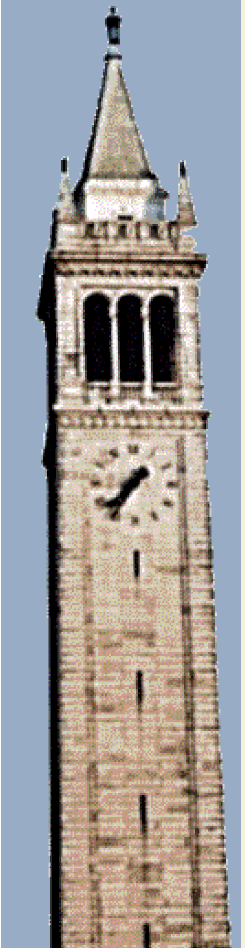
- Use formal models of computation
  - Having a MoC is better than unstructured interaction
  - Having a formal MoC is better than rules of thumb
- Use hierarchies to integrate heterogeneity
- Understand compositionality
  - Precise reactions
  - Behavior type system
- Develop composable models
  - Responsible frameworks

Precise Reaction

- A ***precise reaction*** is a finite piece of computation depends solely on its trigger and leads to a well-defined state.
- A ***compositional precise reaction*** leads a composite actor to a quiescent state.

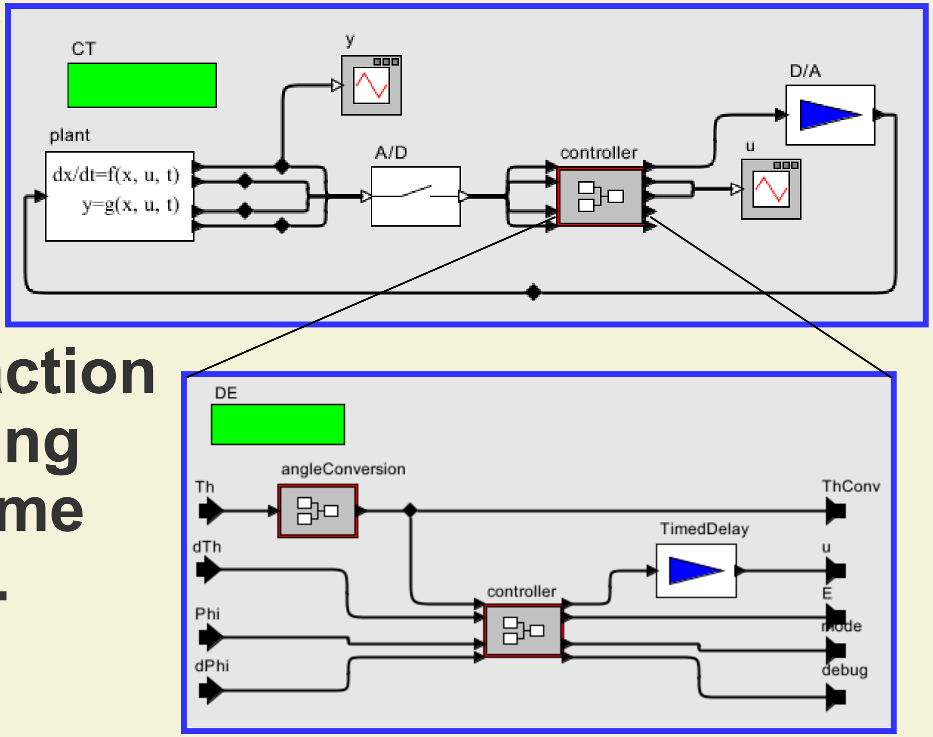




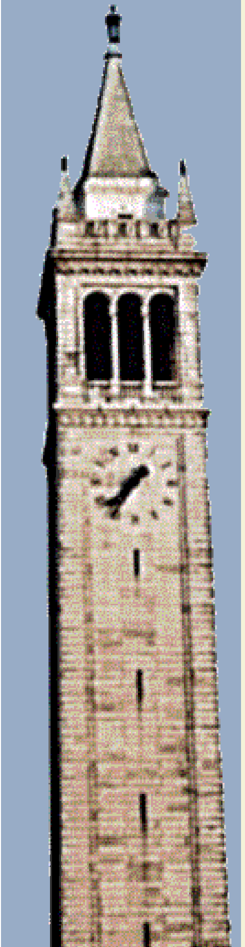


Timed Precise Reaction

- The notion of time provides well-defined *states* of a model at a set of time points.
  - CT: real time line.
  - DE: event time points
  - DT: sampling time
  - SR: tick time
  - …
- The notion of time transfers precise reaction problem into managing the progression of time across models/tools.

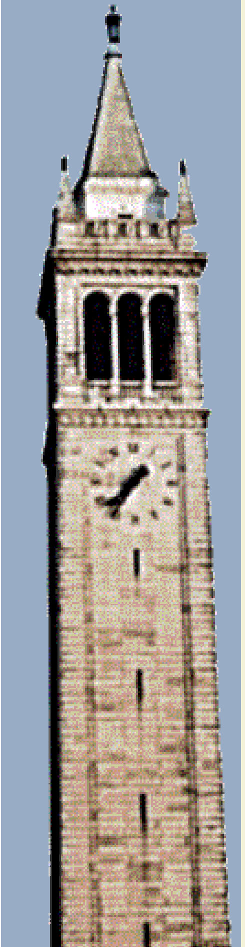


Responsible Frameworks



- A framework implements a model of computation.
- A responsible framework only sends responsible triggers, thus provides compositional precise reaction.
- Not all models of computation have well-defined notion of reaction.
  - communicating sequential processes
  - process network
  - unmanaged prioritized threads
- Not all frameworks are implemented as responsible frameworks.
  - Tools may not support step-by-step execution
  - A “step” may not be a precise reaction

Timed Multitasking
— A responsible real-time framework



- A run-time framework that preserve specified real-time properties.
  - Actors are tasks with finite execution time (not WCET)
    - Tasks are either nonpreemptable or arbitrarily preemptable.
  - Actors specify deadline and priority
    - can cooperate with other tools for schedulability analysis
  - Event-based firing rules are responsible triggers.
  - Split-phase execution and over-run handling to guarantee timing properties.
    - Every actor gets it declared execution time before deadline.
    - If an actor misses its deadline, an overrun-handler will be invoked to bring it to a quiescent state
- Ongoing work: develop TM run time on embedded systems