Process-Based Software Components

Mobies Phase 1, UC Berkeley

Edward A. Lee and Tom Henzinger

PI Meeting, New York

July 24, 2002

PI: Edward A. Lee, 510-642-0455, eal@eecs.berkeley.edu
Co-PI: Tom Henzinger, 510-643-2430, tah@eecs.berkeley.edu
PM: John Bay
Agent: James Lyttle, AFRL/IFSC, James.Lyttle@wpafb.af.mil
Award end date: December, 2003
Contract number: F33615-00-C-1703
AO #: J655

Subcontractors and Collaborators

- Subcontractor
  - Univ. of Maryland (C code generation)
- Collaborators
  - UCB Phase II
  - Kestrel
  - Vanderbilt
  - Penn
- Non-Mobies
  - GSRC project (system-level IC design)
  - SEC program (Boeing, etc.)

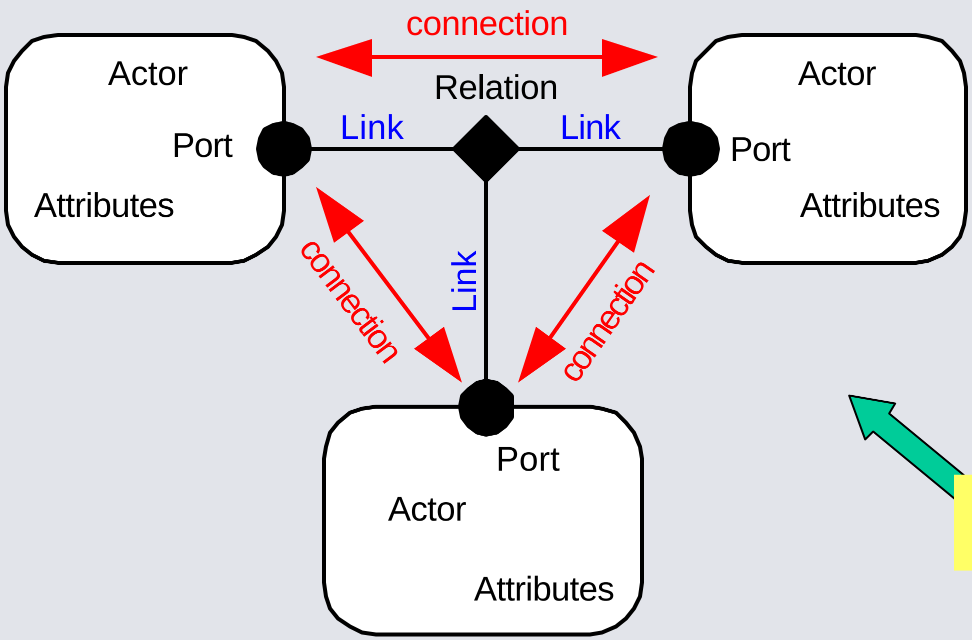
Program Objectives

Our focus is on component-based design using principled *models of computation* and their *runtime environments* for embedded systems. The emphasis of this project is on the dynamics of the components, including the communication protocols that they use to interface with other components, the modeling of their state, and their flow of control. The purpose of the mechanisms we develop is to improve robustness and safety while promoting component-based design.

Technical Approach Summary

- Models of computation
  - supporting heterogeneity
  - supporting real-time computation
  - codifications of design patterns
  - definition as *behavioral types*
- Co-compilation
  - joint compilation of components and architecture
  - vs. code generation
  - supporting heterogeneity
- Ptolemy II ← our tool
  - our open-architecture software laboratory
  - shed light on models of computation & co-compilation
  - by prototyping modeling frameworks and techniques

View of Concurrent Components: *Actors* with *Ports* and *Attributes*



Model of Computation:

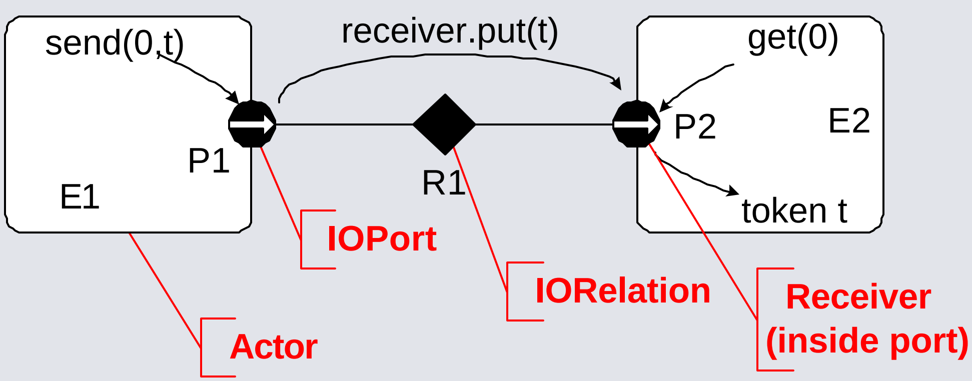
- Messaging schema
- Flow of control
- Concurrency

our basic meta model

Key idea: The model of computation is part of the framework within which components are embedded not part of the components themselves. It enforces patterns.

Actor-Oriented View of Producer/Consumer Components

Basic Transport:



Models of Computation:

- continuous-time
- dataflow
- rendezvous
- discrete events
- synchronous
- time-driven
- publish/subscribe
- push/pull
- …

our meta model for producer/consumer models of computation

Examples of Actor-Oriented Component Frameworks

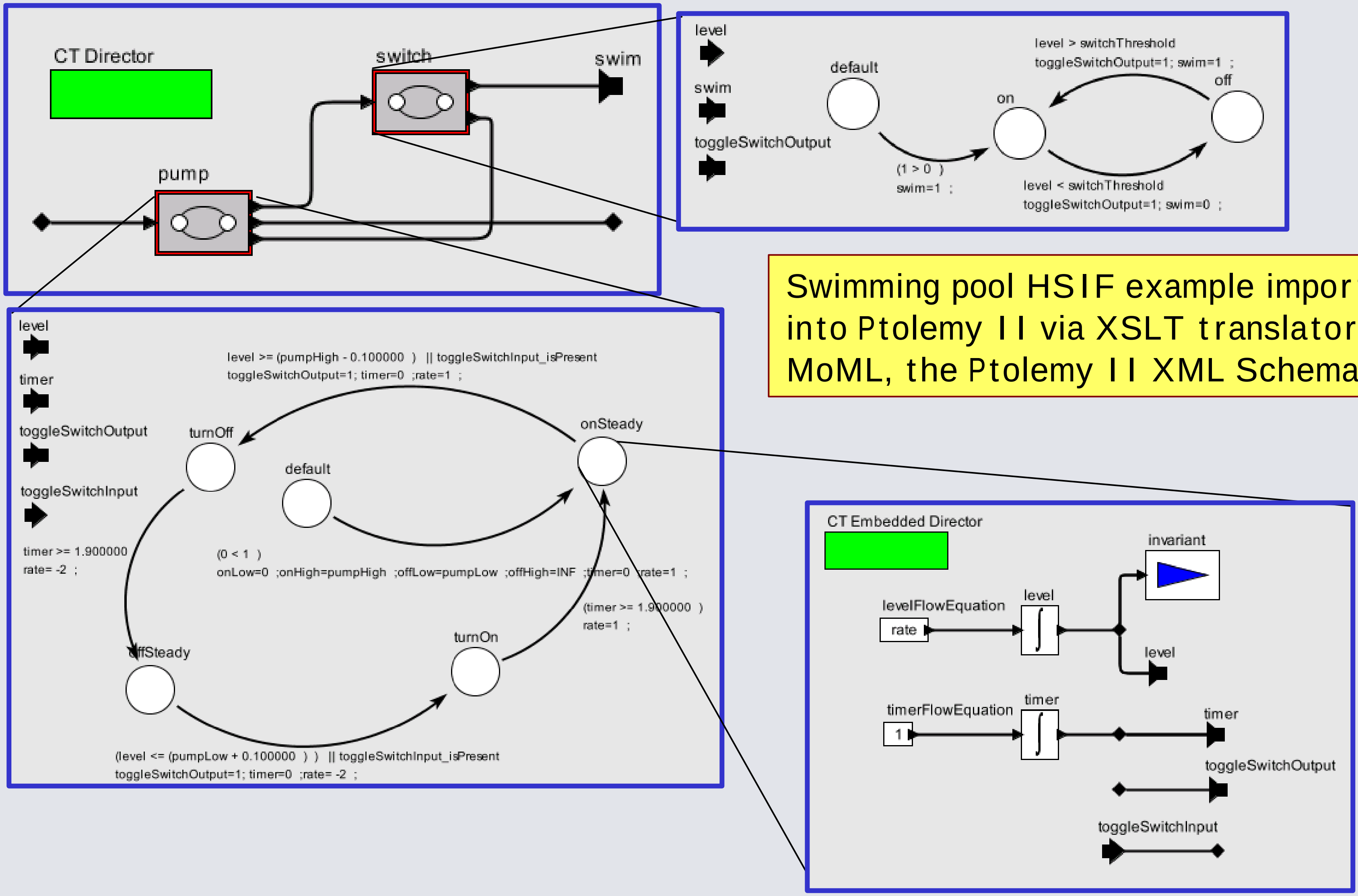
- Simulink (The MathWorks)
- Labview (National Instruments)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- VHDL, Verilog, SystemC (Various)
- Polis & Metropolis (UC Berkeley)
- Ptolemy & Ptolemy II (UC Berkeley)
- ...

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
    - Stateflow – finite-state machines
    - 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

Part I: Networks of Automata

- Heterogeneous Mixtures of Automata and Actors -



Swimming pool HSIF example imported into Ptolemy II via XSLT translator to MoML, the Ptolemy II XML Schema.

Semantics Questions

- What automata can be expressed?
  - nondeterministic, guard expression language, actions, …
- How are transitions in distinct automata coordinated?
  - synchronous, time-driven, event-driven, dataflow, …
  - can outputs and updates be separated?
- What can automata communicate?
  - messages, events, triggers
- How communications carried out?
  - synchronous, rendezvous, buffered, lossy, …
- How are continuous variables shared?
  - global name space, scoping, mutual exclusion, …
- What is the meaning of directed cycles?
  - fixed point, error, infinite loop, …
- What is the meaning of simultaneous events?
  - secondary orderings, such as data precedences, priorities, …

Possible Interaction Semantics Between Automata

- Asynchronous
  - Promela (specification language for Spin)
  - SDL
  - Ptolemy II (PN+FSM, DE+FSM)
- Synchronous w/ fixed point
  - Esterel
  - Simulink
  - Ptolemy II (SR+FSM)
- Synchronous w/out fixed point
  - Giotto
  - Ptolemy II (SDF+FSM)
  - HSIF

Tool Integration Efforts

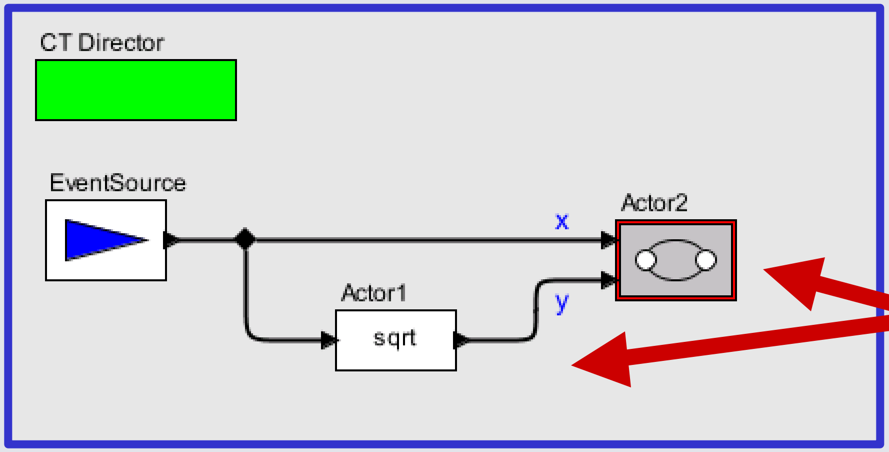
- First attempt: Charon
  - Created a Charon parser (in Java) to import into Ptolemy II CT+FSM domains
  - Created a Charon code generator to produce Charon from Ptolemy II CT+FSM models
- Second attempt: HSIF
  - Created an XSLT translator from HSIF to MoML, creating Ptolemy II models in CT+FSM.
  - This is ongoing, as we are resolving semantics mismatches.

Difficulties

- CT+FSM does not model the two sources of nondeterminism in HSIF:
  - At each time step, we order execution of the automata according to data precedences, as done by Simulink, for example.
    We believe this is a flaw in HSIF
  - The FSM domain rejects nondeterminism where more than one guard is enabled at a state.
    We believe this is a flaw in the FSM domain of Ptolemy II, but:

    What should a simulator do with such nondetermism? It is incorrect to just choose one of the enabled transitions (because it will lead to untrustable simulations).
- Several other difficulties with global variables and directed cycles in HSIF.

Order of Execution Question

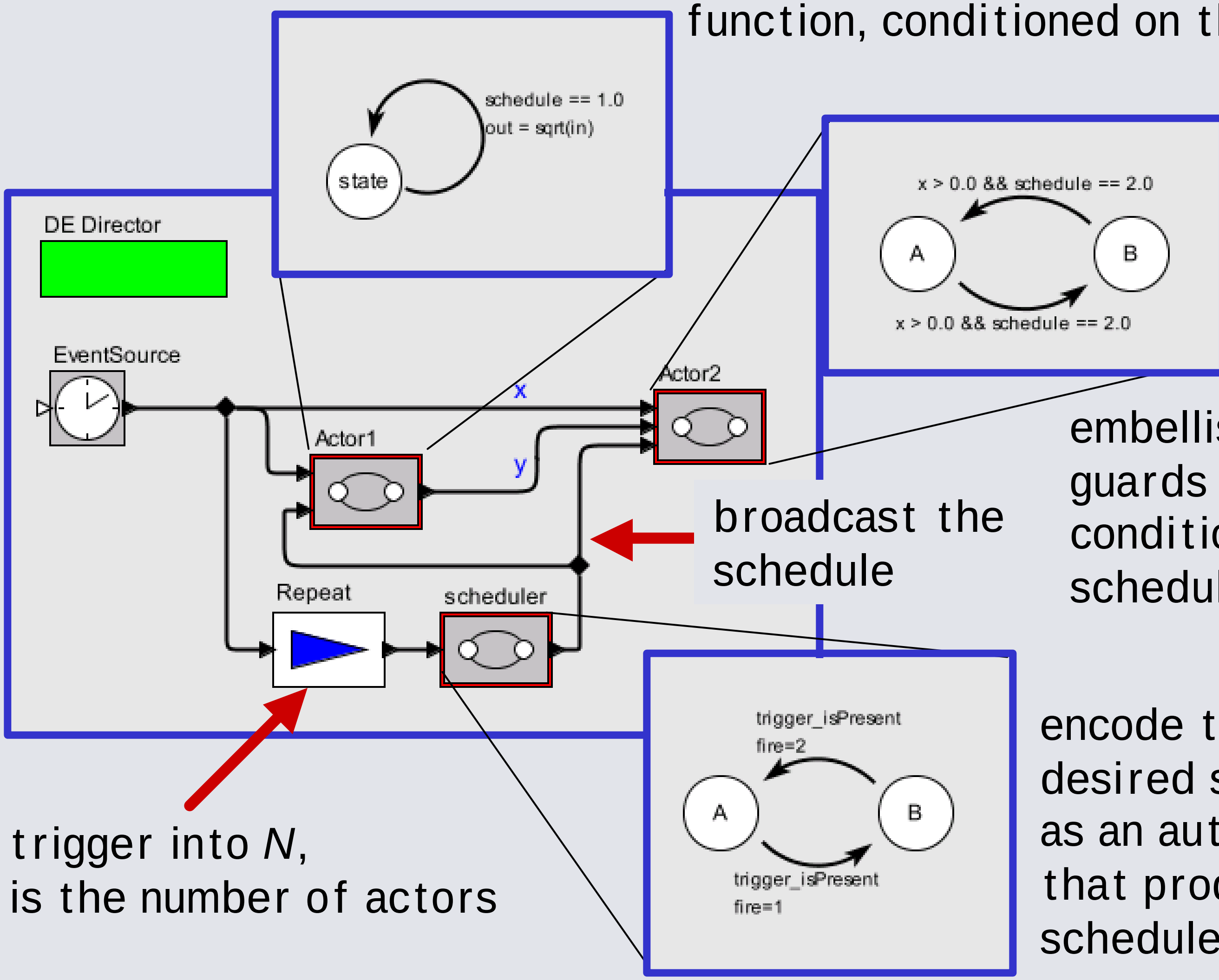


Given an event from the event source, which of these should react first? HSIF declares this to be nondeterministic.

Simulink and the Ptolemy II CT domain declare this to be deterministic, based on data precedences. Actor1 executes before Actor2.

Using HSIF Semantics to Get Determinism is Hard

conditionally execute the simple function, conditioned on the schedule



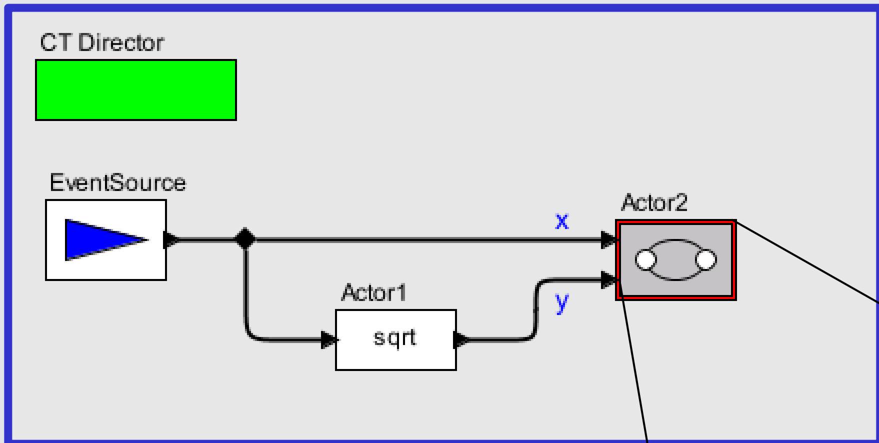
broadcast the schedule

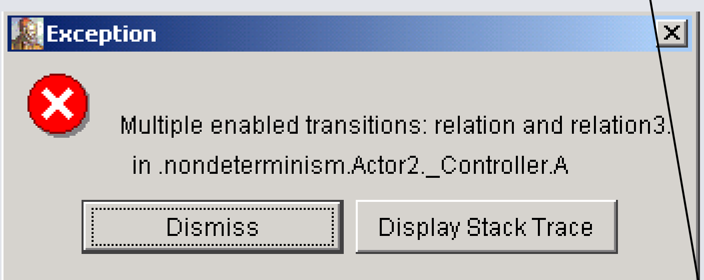
embellish the guards with conditions on the schedule

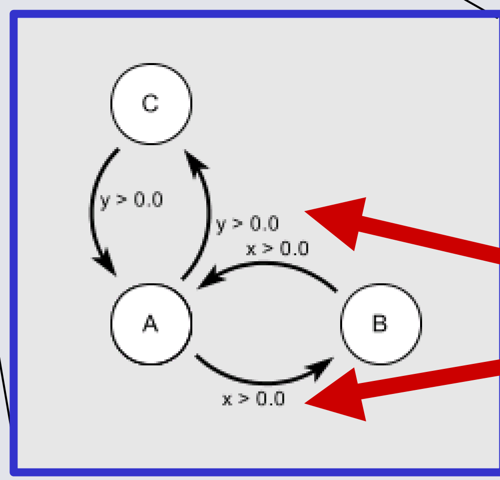
turn one trigger into *N*, where *N* is the number of actors

encode the desired sequence as an automaton that produces a schedule

Using CT or Simulink Semantics to Get Nondeterminism is Easy







Ptolemy's FSM domain throws an exception upon encountering such nondeterminism. Stateflow uses the position of transitions to disambiguate. Neither of these correctly reflects HSIF semantics.

At a time when the event source yields a positive number, both transitions are enabled.

HSIF Premise

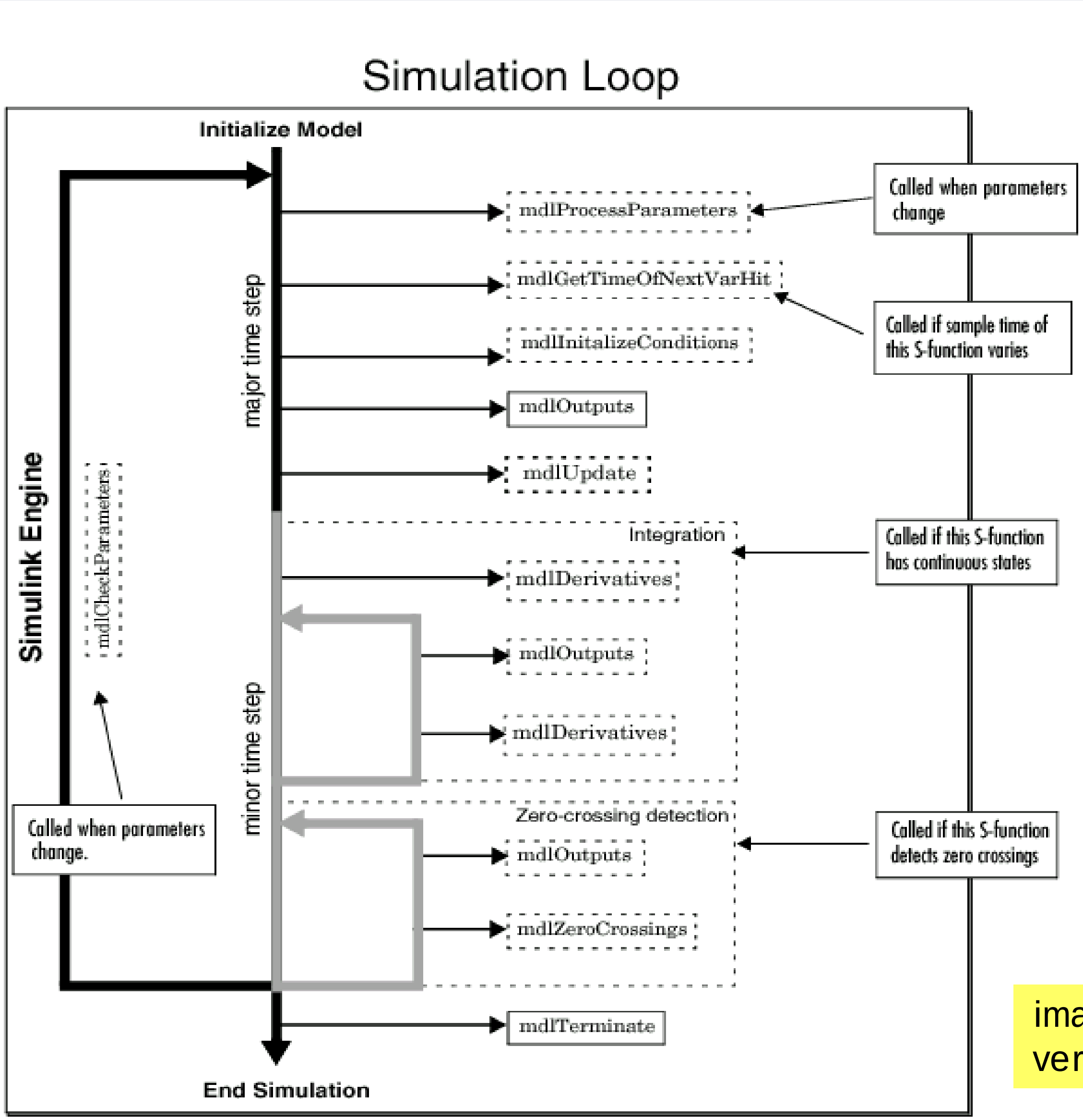
- Time as a binding agent makes sense.
  - Because of time-based differential equations.
  - Hence, rule out asynchronous semantics.
- Global variables for sharing continuous signals
  - How to enforce or analyze mutual exclusion?
- Nondeterminate
  - Automata can have multiple enabled transitions
  - Simultaneous triggers yield nondeterminate ordering of reactions

We quibble with the latter:
- It surprises the designer
- It is hard to get determinism when this is desired
- Getting the desired nondeterminism is easy using the former
- Writing simulators that are trustworthy is difficult
  - It is incorrect to just pick one possible behavior!
- With this semantics, it makes no sense to export to HSIF from Ptolemy II CT+FSM models or from Simulink+Stateflow models.

Part II: Real-Time Actor Semantics

- Simulink
  - underlying continuous-time semantics
  - good support for periodic real-time tasks
  - code generation via real-time workshop
- Giotto
  - underlying time-triggered semantics
  - execution on embedded machine
- Timed Multitasking (TM)
  - reactive, aperiodic semantics
  - delayed output commit, as in Giotto
- Others:
  - Real-time CORBA
  - Port-based objects (PBO)
  - Timed process networks
  - Timed CSP

Simulink Semantics

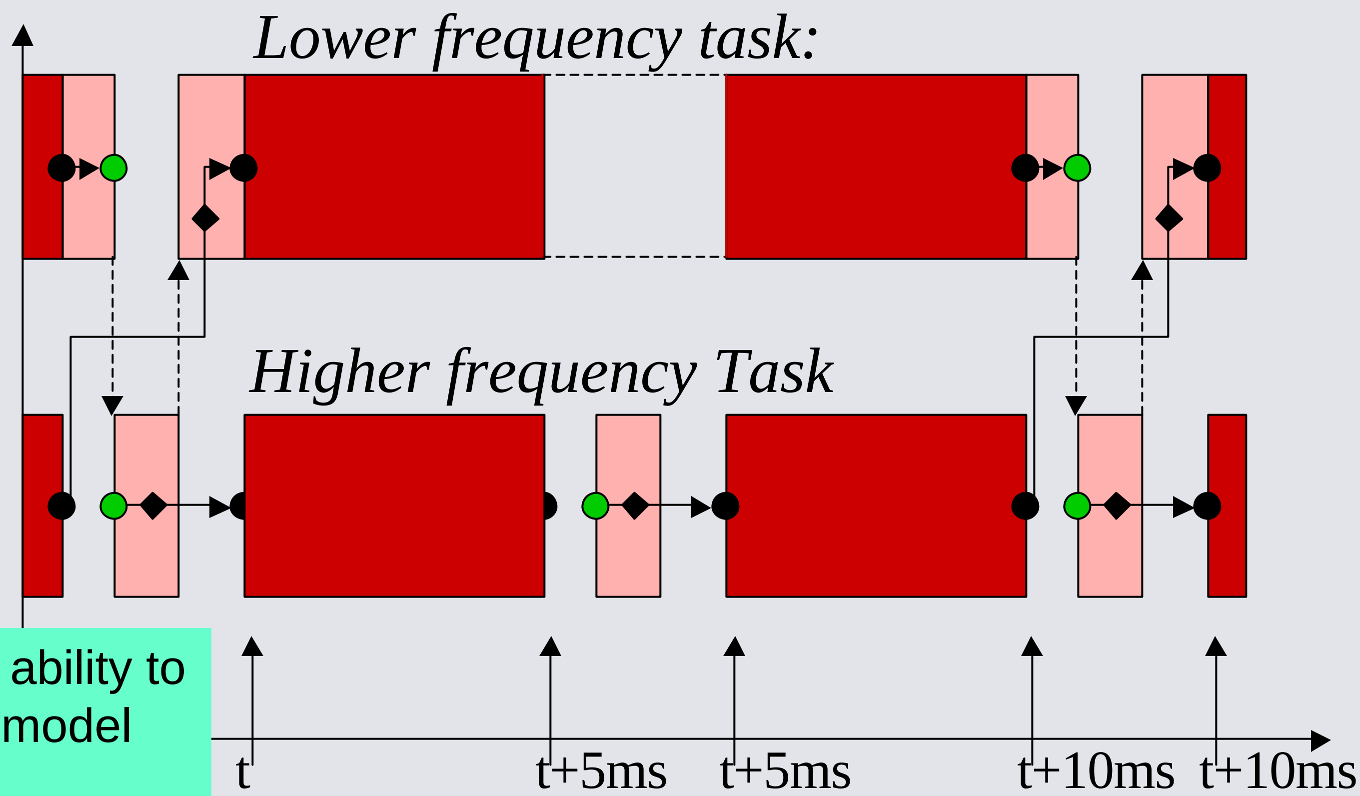


- continuous time
- discrete actors are logically instantaneous
- separation of output/update methods to support algebraic loops, integration, and zero-crossing detection
- output method invoked many times
- multitasking mode for periodic discrete-time tasks.
- multitasking mode requires Giotto-like delayed output commit

image from *Writing S-Functions*, version 4, The MathWorks

Giotto – Periodic Hard-Real-Time Tasks with Precise Mode Changes

Major part of the Mobies effort is to interface this domain to others: CT above, FSM below for modal modeling, and SDF for task definition.



Task 1.2: Demonstrate ability to model domain specific model semantics

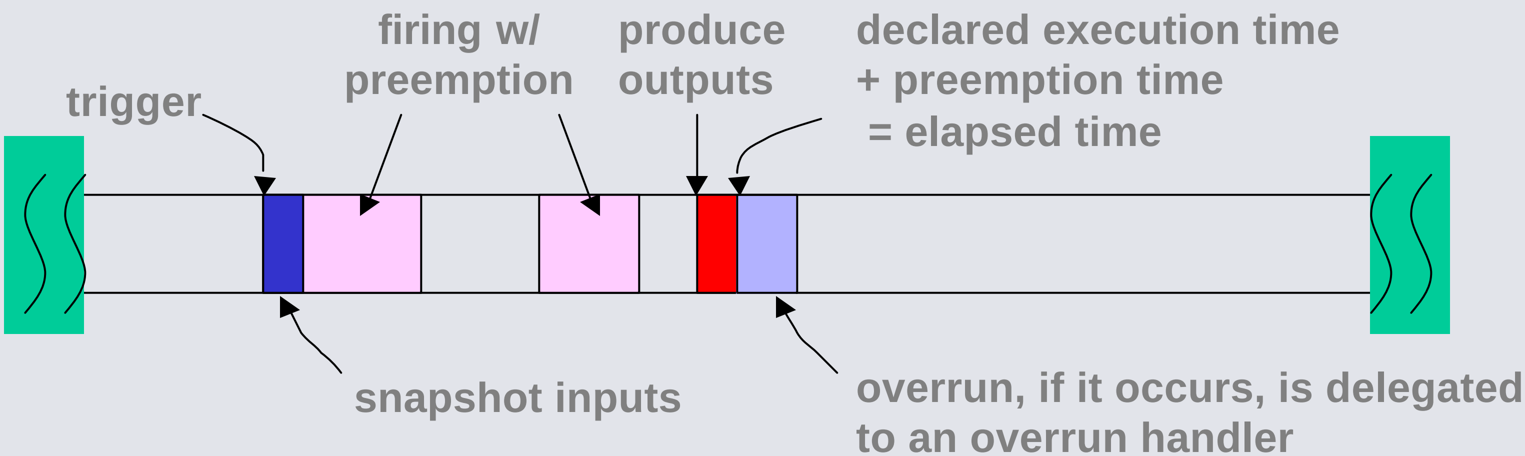
- Giotto compiler targets the E Machine
- Giotto/Simulink integration
- Ptolemy II Giotto code generator

Timed Multitasking (TM)

- Extending this Concept to Event-Driven Models -

- Actors are triggered by input events
- Snapshot of inputs upon triggering
- Concurrent execution according to priorities
- $t_o = t_i + T + P$
  - t_o = Time of outputs
  - t_i = Time of inputs
  - T = declared execution time
  - P = preemption time

Task 1.1: Demonstrate ability of modeling cross cutting physical constraints

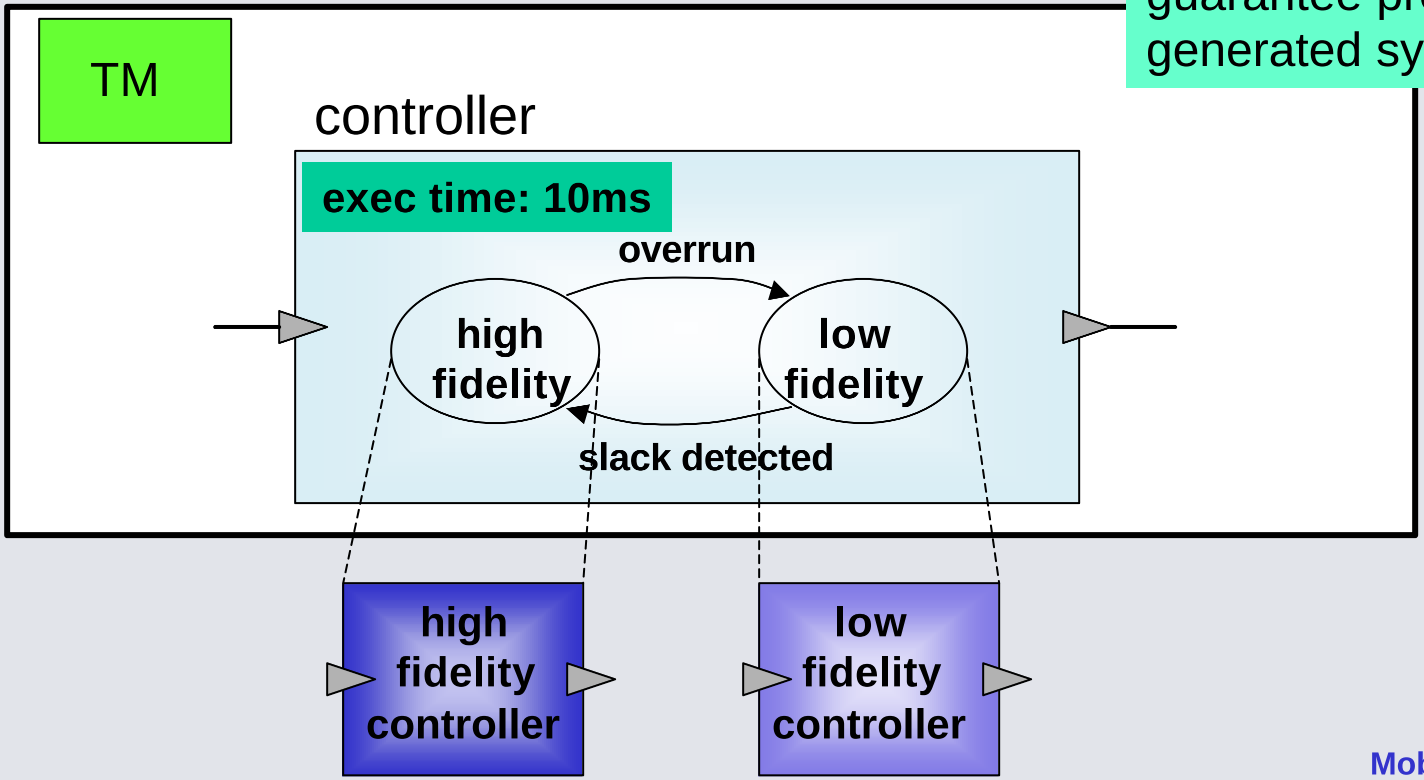


Overrun Handlers in TM

When overrun occurs, handler can:

- commit partial results (for *anytime* algorithms)
- roll back (*transaction* semantics)
- switch to degraded mode operation;
- suspend rogue tasks
- raise an alarm

2.7 Demonstrate ability to guarantee properties of generated systems



Comparisons

- Giotto, Simulink, and TM, all achieve data determinism with snapshot of inputs and delayed commit of outputs.
- Giotto introduces a unit delay in any communication. Simulink introduces a unit delay only on sample rate changes. TM does not introduce a unit delay.
- Simulink requires output/update separation. The others do not.
- TM builds in the notion of an *overrun handler*. The others do not.
- *Common principles:*
  - responsible frameworks
  - precise reactions

Abstract Semantics

The "right" abstract semantics would allow these models of computation to be composed with one another and with other MoCs. This abstract semantics will have:

- output/update separation
  - required by Simulink
- finite actor computation
  - required by all
- predictable execution times
  - required by all
- declared execution times
  - required by TM, and by Giotto and Simulink for schedulability analysis.

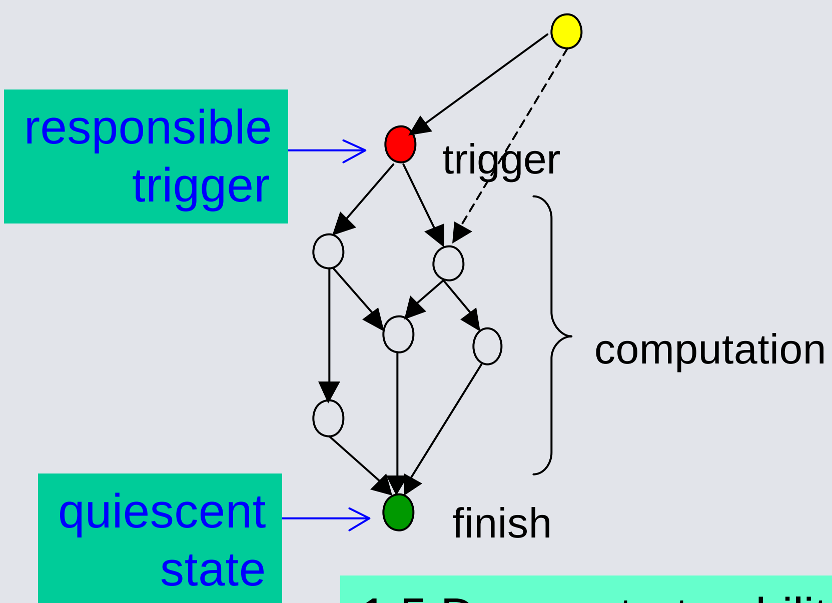
1.5 Demonstrate ability to integrate different models of concurrency

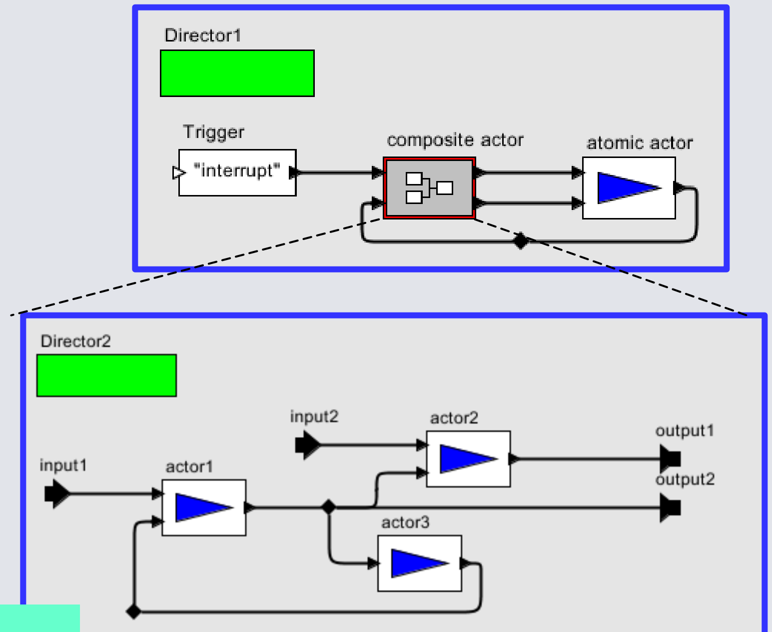
1.6 Demonstrate ability to integrate domain specific modeling tools

1.7 Demonstrate ability to compose multiple view models

A Theory of Responsible Frameworks
- Ensures Finite Actor Computation -

- 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.
- A *responsible framework* only sends responsible triggers, thus guarantees compositional precise reaction.





1.5 Demonstrate ability to integrate different models of concurrency

Status update: Code Generation

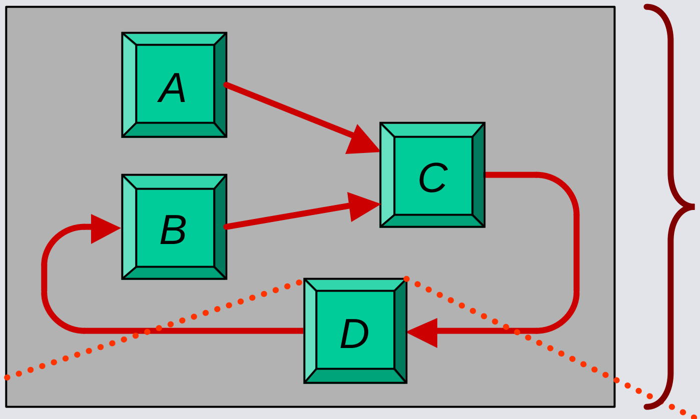
- It is not sufficient to build a mechanism for generating code from one, fixed, modeling environment.
- Modeling strategies must be nested hierarchically.
- Code generators have to be heterogeneously composable.

Task 2.3: Demonstrate ability to compose generators from components

Code Generation Status

- Giotto code generator from Giotto domain
  - still need code generation from FSM to get modal models
- Java code generator from SDF domain
  - based on Soot compiler infrastructure (McGill)
  - type specialization
  - static scheduling, buffering
  - code substitution using model of computation semantics
- C code generation from Java
  - University of Maryland subcontract
  - based on Soot compiler infrastructure (McGill)
  - preliminary concept demonstration built
- Configurable hardware synthesis
  - targeted Wildcard as a concept demonstration
  - collaborative with BYU (funded by another program)

Actor Definition Status Update



Code generate a domain-polymorophic component definition.

```
public TypedIOPort input;
public TypedIOPort output;
public Parameter constant;
public void fire() {
        Token t = input.get(0);
        Token sum = t.add(constant.getToken());
        output.send(0, t2);
}
```

Actor Definition: Cal

- Java is not the ideal actor definition language. Key meta-data is hard to extract:
  - token production/consumption patterns
  - firing rules (preconditions)
  - state management (e.g. recognize stateless actors)
  - type constraints must be explicitly given
  - modal behavior
- Defining an actor definition format (Cal):
  - enforce coding patterns
  - make meta-data available for code generation
  - infer behavioral types automatically
  - analyze domain compatibility
  - support multiple back-ends (C, C++, Java, Matlab)

Summary of Accomplishments to Date

- Heterogeneous modeling
  - Domain polymorphism concept & realization
  - Behavioral type system
  - Giotto semantics & integration with other MoCs
  - Component definition principles (Cal)
- Tool integration
  - Charon import/export from Ptolemy II
  - (partial) HSIF import to Ptolemy II
  - Matlab integration with Ptolemy II
- Code generation
  - Co-compilation concept
  - Giotto program generation
  - Java code generation from SDF
  - C code generation from Java
    - Early phase, concept demonstration

Plans

- OEP
  - ETC and V2V models and code generators
- HSIF
  - Resolve semantics questions and create Ptolemy II interface
- Complete actor definition framework
  - define the meta-semantics for domain-polymorphic actors
- Behavioral types
  - support reflection
  - real-time properties as dependent types
- Complete SDF code generation
  - token unboxing
  - elimination of memory management
  - 100% of test suite must pass

More Plans

- Code generate Ptolemy II expressions
  - use of expression actor simplifies models
  - expressions for guards and actions in FSMs
- Implement FSM code generation
  - support modal models
- Complete C code generation
  - support key subset of Java libraries
- Integrate heterogeneous code generators
  - systematize hierarchy support
  - define Java subset that generates well to C
- Investigate Simulink/Ptolemy II interaction
  - focus on the abstract semantics