Overview of the Ptolemy Project

Edward A. Lee
Robert S. Pepper Distinguished Professor and
Chair of EECS, UC Berkeley

EECS 249 Guest Lecture

Berkeley, CA
September 20, 2007

Elevator Speech

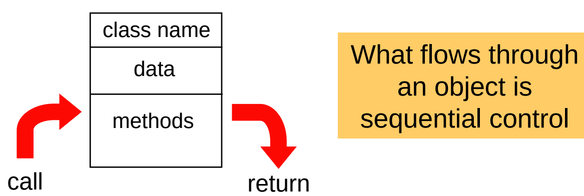
The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined models of computation that govern the interaction between components. A major problem area being addressed is the use of heterogeneous mixtures of models of computation. A software system called Ptolemy II is being constructed in Java, and serves as the principal laboratory for experimentation.

Concurrent Composition of Subsystems, In Mainstream SW Engineering in 2007

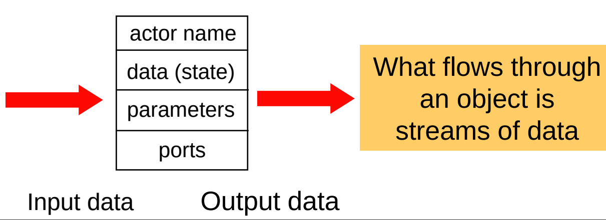
- Component technologies
  - Objects in C++, C#, or Java
  - Wrappers as service definitions
- Concurrency
  - Threads (shared memory, semaphores, mutexes, …)
  - Message Passing (synchronous or not, buffered, …)
- Distributed computing
  - Distributed objects wrapped in web services, Soap, CORBA, DCOM, …

Our Approach: *Actor-Oriented Models of Computation*

Traditional component interactions:



Actor oriented:



Software Legacy of the Project

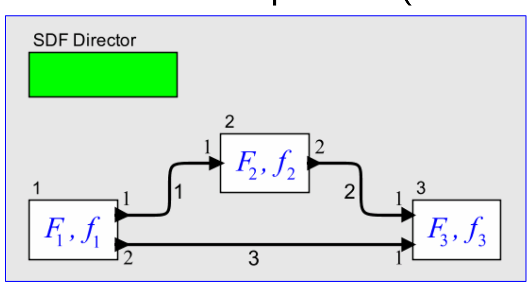
- Gabriel (1986-1991)
  - Written in Lisp
  - Aimed at signal processing
  - Synchronous dataflow (SDF) block diagrams
  - Parallel schedulers
  - Code generators for DSPs
  - Hardware/software co-simulators
- Ptolemy Classic (1990-1997)
  - Written in C++
  - Abstract Actor Semantics
  - Multiple models of computation
  - Hierarchical heterogeneity
  - Dataflow variants: BDF, DDF, PN
  - C/VHDL/DSP code generators
  - Optimizing SDF schedulers
  - Higher-order components
- Ptolemy II (1996-2022)
  - Written in Java
  - Behavioral polymorphism
  - Multithreaded
  - Network integrated and distributed
  - Modal models
  - Sophisticated type system
  - CT, HDF, CI, GR, etc.

Each of these served us, first-and-foremost, as a laboratory for investigating design.

Focus has always been on system modeling and embedded software.

Lee, Berkeley 5

Where it started: SDF: Synchronous Dataflow and the Balance Equations (1985-86)



production/consumption matrix

Connector 1

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$

Actor 1

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

firing vector

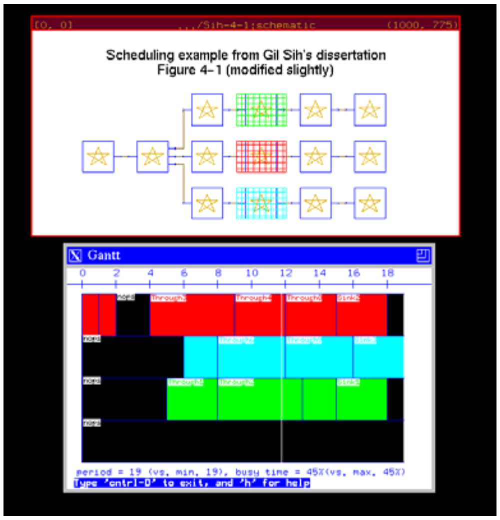
balance equations

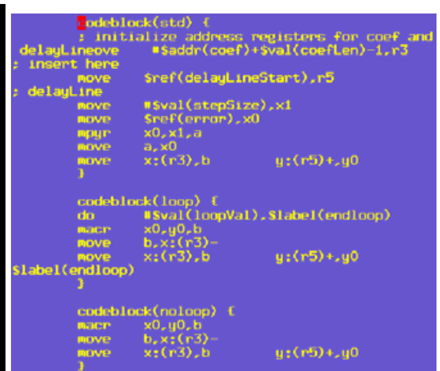
$$\Gamma q = \vec{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Lee, Berkeley 6

Gabriel and Ptolemy Classic Leveraged SDF to Generate Parallel Code

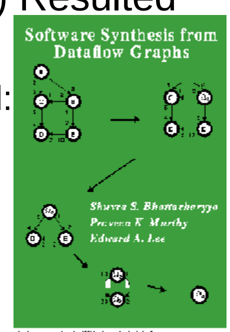
SDF model, parallel schedule, and synthesized assembly code (1990)





It is an interesting (and rich) research problem to minimize interlocks in complex multirate applications.

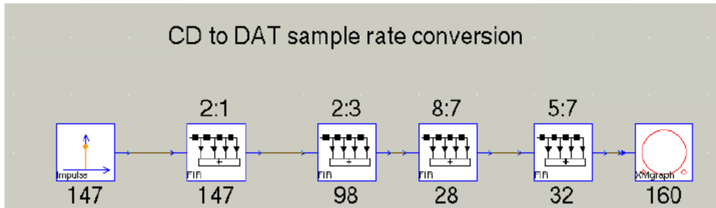
Many Scheduling and Optimization Problems (and Some Solutions) Resulted



- Optimization criteria that might be applied:
- Minimize buffer sizes.
- Minimize the number of actor activations.
- Minimize the size of the representation of the schedule (code size).

See S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs,* Kluwer Academic Press, 1996, for a summary of the single processor optimization problems.

Minimum Buffer Schedule

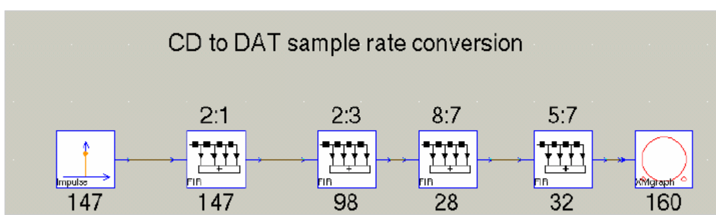


A B A B C A B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C D E
A F F F F F B C A B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C
D E A F F F F F B A B C A B C A B A B C A B C D E A F F F F F B A B C A B C A
B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F F F F F E B C A
F F F F F B A B C A B C D E A F F F F F B A B C A B C A B A B C A B C D E A F
F F F F B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C A B A B C
D E A F F F F F B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C A
B C D E A F F F F F B A B C A B C A B A B C D E A F F F F F E B C A F F F F F B
A B C A B C A B A B C D E A F F F F F B C A B A B C A B C D E A F F F F F B A
B C A B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C D E A F F F
F F B C A B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C D E A F
F F F F B A B C A B C A B A B C A B C D E A F F F F F E B A F F F F F B C A B C
A B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F F F F F B C A
B A B C A B C D E A F F F F F B A B C A B C A B A B C A B C D E A F F F F F B
A B C A B C A B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F
F F F F B C A B A B C A B C D E F F F F F F E F F F F F

Source: Shuvra Bhattacharyya

Scheduling Tradeoffs

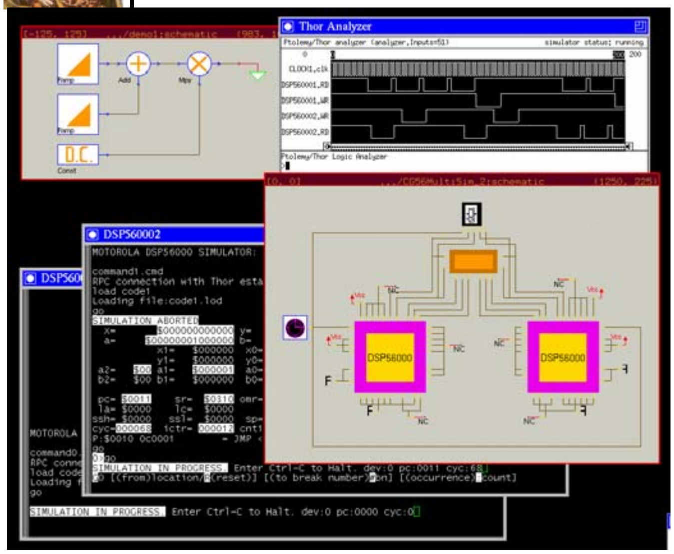
(Bhattacharyya, Parks, Pino, Lee)



Scheduling strategy	Code	Data
Minimum buffer schedule, no looping	13735	32
Minimum buffer schedule, with looping	9400	32
Worst minimum code size schedule	170	1021
Best minimum code size schedule	170	264

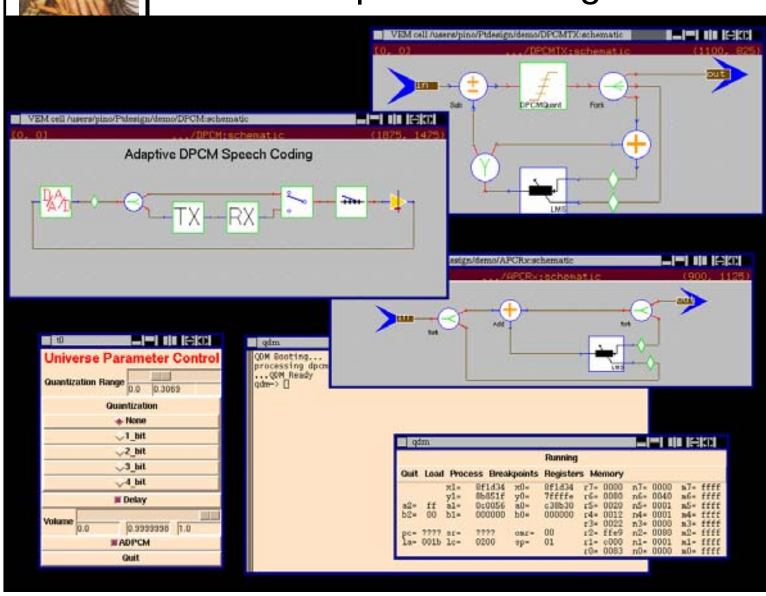
Source: Shuvra Bhattacharyya

Gabriel and Ptolemy Classic Provided Co-Simulation of Hardware and Generated Software

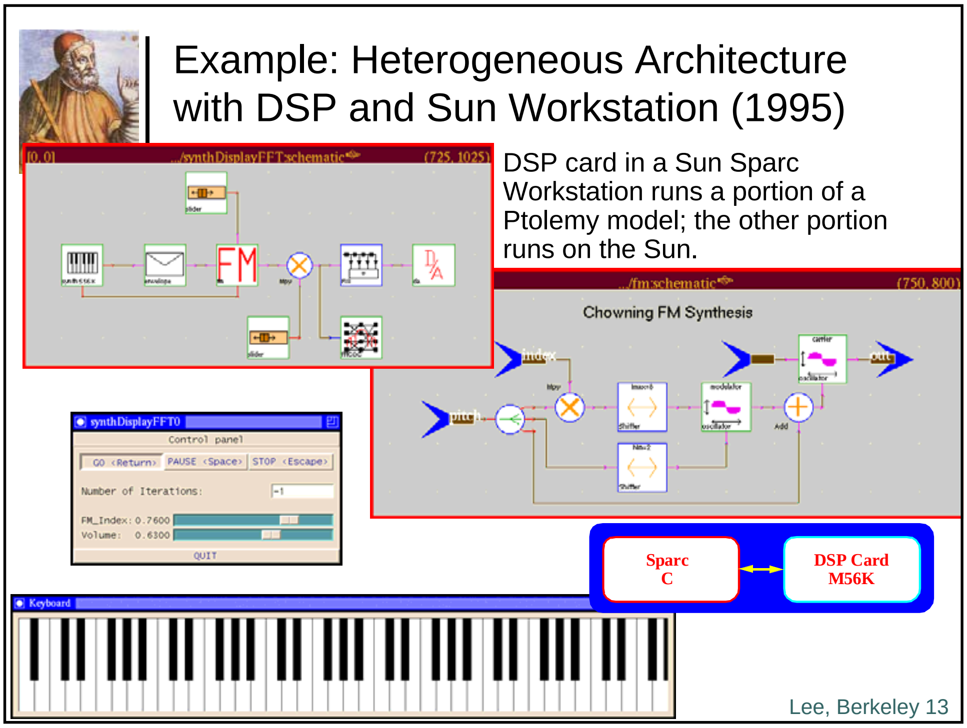


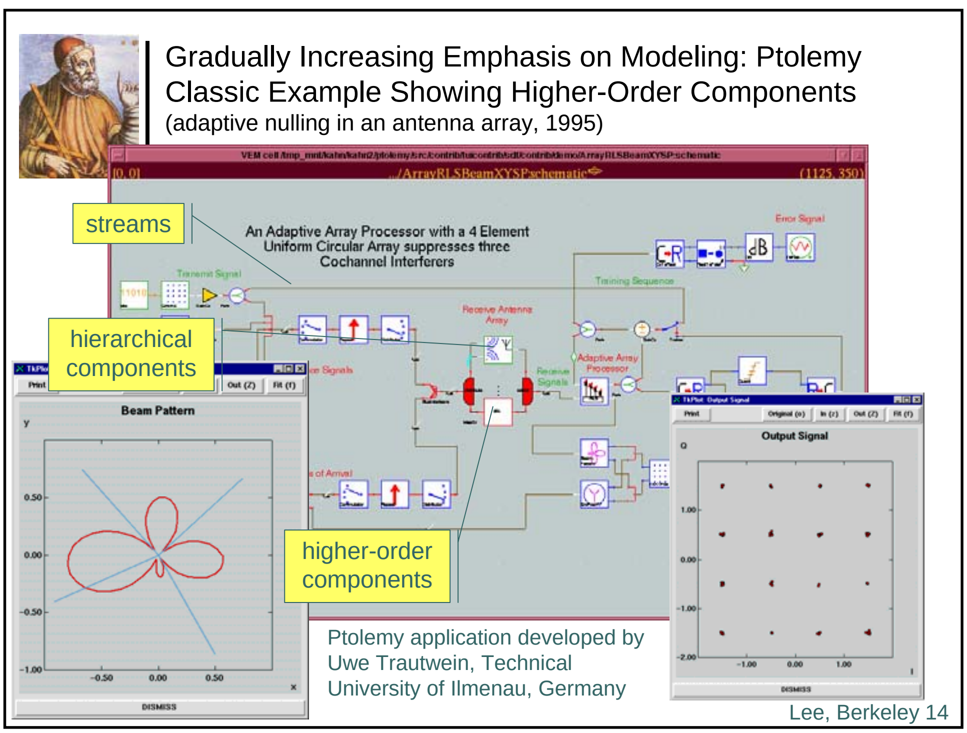
An SDF model, a "Thor" model of a 2-DSP architecture, a "logic analyzer" trace of the execution of the architecture, and two DSP code debugger windows, one for each processor (1990).

Example of Model-Based Design: ADPCM Speech Coding



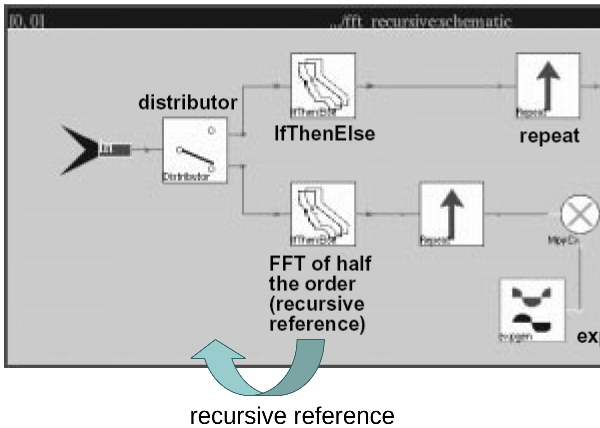
Model of a speech coder generated to DSP assembly code and executed using a DSP debugger interface with host/DSP interaction (1993).

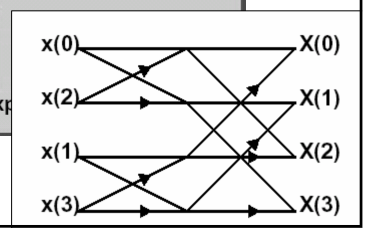

Example: Heterogeneous Architecture with DSP and Sun Workstation (1995)
DSP card in a Sun Sparc Workstation runs a portion of a Ptolemy model; the other portion runs on the Sun.
Chowning FM Synthesis
synthDisplayFFT0
Control panel
GO ‹Return›
PAUSE ‹Space›
STOP ‹Escape›
Number of Iterations:
FM_Index: 0.7600
Volume: 0.6300
QUIT
Keyboard
Sparc C
DSP Card M56K
Lee, Berkeley 13


Gradually Increasing Emphasis on Modeling: Ptolemy Classic Example Showing Higher-Order Components
(adaptive nulling in an antenna array, 1995)
streams
hierarchical components
higher-order components
An Adaptive Array Processor with a 4 Element Uniform Circular Array suppresses three Cochannel Interferers
Beam Pattern
Output Signal
DISMISS
Ptolemy application developed by Uwe Trautwein, Technical University of Ilmenau, Germany
Lee, Berkeley 14

Higher-Order Components Realizing Recursion in Ptolemy Classic

FFT implementation in Ptolemy Classic (1995) used a partial evaluation strategy on higher-order components.

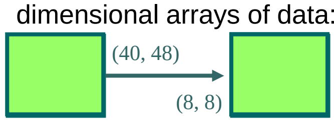




Extension of SDF: Multidimensional SSDF

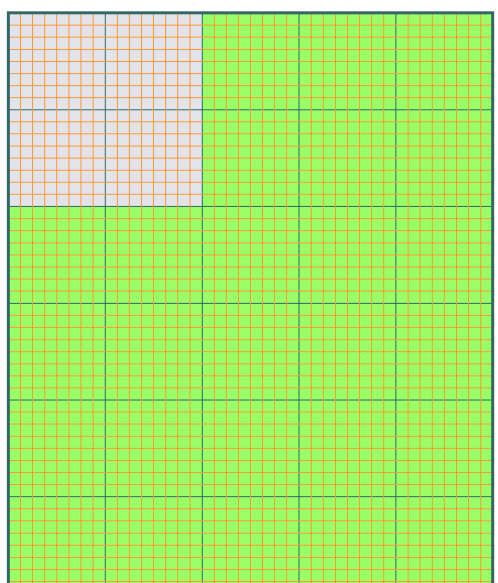
(1993)

- Production and consumption of *N*-dimensional arrays of data:



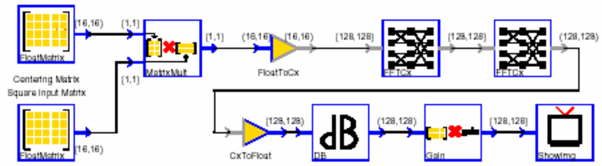
- Balance equations and scheduling policies generalize.
- Much more data parallelism is exposed.

Similar (but dynamic) multidimensional streams have been implemented in Lucid.



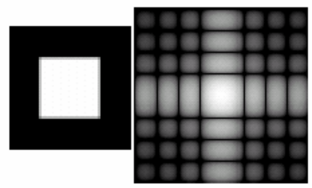
More interesting Example

Two dimensional FFT constructed out of one-dimensional actors.

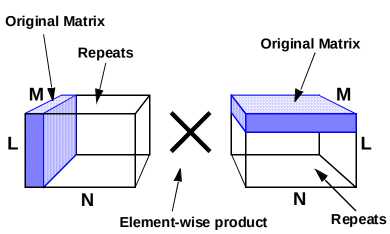


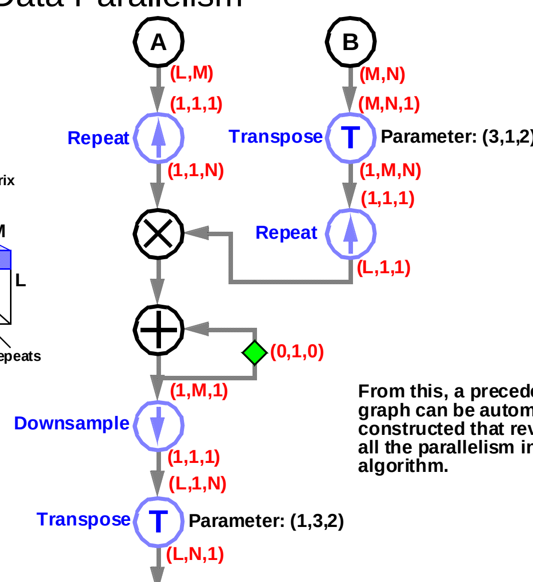

```
MDSDF SCHEDULE:
fft_of_square2.FloatMatrix1, firing range: (0,0)
fft_of_square2.FloatMatrix2, firing range: (0,0)
fft_of_square2.Mult1, firing range: (0,0) - (15,15)
fft_of_square2.FloatToCx1, firing range: (0,0)
fft_of_square2.FFTCx2, firing range: (0,0) - (15,0)
fft_of_square2.FFTCx1, firing range: (0,0) - (0,127)
fft_of_square2.CxToFloat1, firing range: (0,0)
fft_of_square2.DB1, firing range: (0,0)
fft_of_square2.Gain1, firing range: (0,0)
fft_of_square2.ShowImg1, firing range: (0,0)
```

Figure 6. Screen dump of 2D-FFT system, the associated schedule, and outputs.



MDSSDF Structure Exposes Fine-Grain Data Parallelism





From this, a precedence graph can be automatically constructed that reveals all the parallelism in the algorithm.

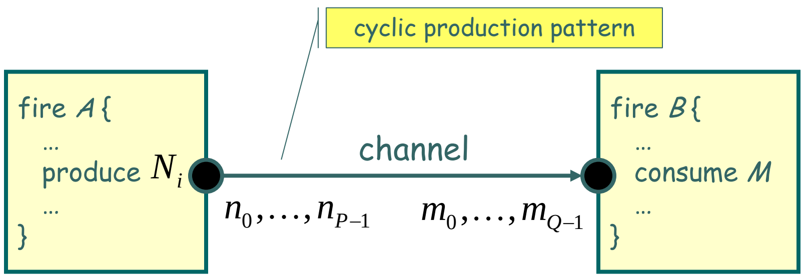
However, such programs are extremely hard to write (and to read).

Lee, Berkeley 18

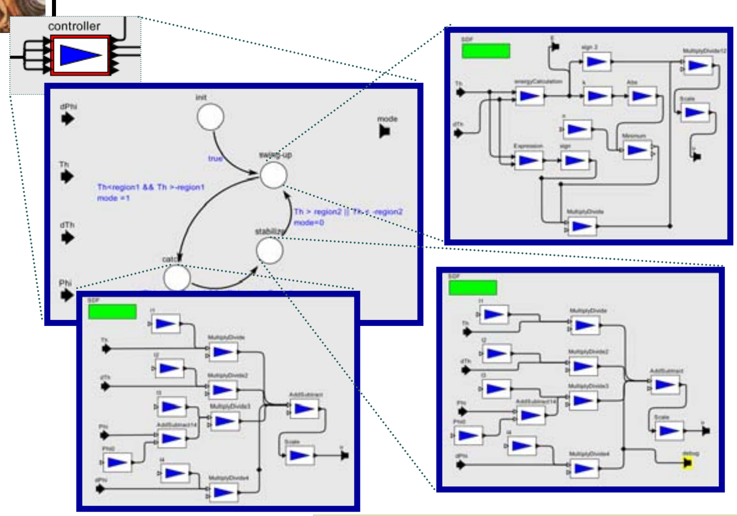
Another Extension: Cyclostatic Dataflow (CSDF) Lauwereins et al., TU Leuven, 1994)

- Actors cycle through a regular production/consumption pattern.
- Balance equations become:

$$q_A \sum_{i=0}^{R-1} n_{i \bmod P} = q_B \sum_{i=0}^{R-1} m_{i \bmod Q}; \; R = lcm(P, Q)$$



Further Extension: Heterochronous Dataflow (HDF) Girault, Lee, & Lee, 1997)



An actor consists of a state machine and refinements to the states that define behavior.

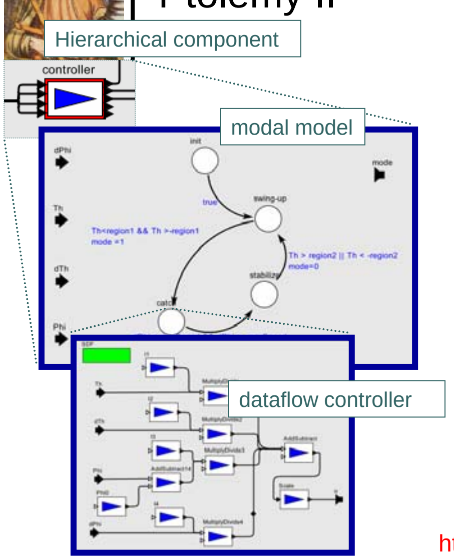
Heterochronous Dataflow (HDF)

(Girault, Lee, and Lee, 1997)

Related to "parameterized dataflow" of Bhattacharya and Bhattacharyya (2001).

- An interconnection of actors.
- An actor is either SDF or HDF.
- If HDF, then the actor has:
  - a state machine
  - a refinement for each state
  - where the refinement is an SDF or HDF actor
- Operational semantics:
  - with the state of each state machine fixed, graph is SDF
  - in the initial state, execute one complete SDF iteration
  - evaluate guards and allow state transitions
  - in the new state, execute one complete SDF iteration
- HDF is decidable if state machines are finite
  - but complexity can be high

Ptolemy II



example Ptolemy II model: hybrid control system

Ptolemy II:

Our current framework for experimentation with actor-oriented design, concurrent semantics, visual syntaxes, and hierarchical, heterogeneous design.

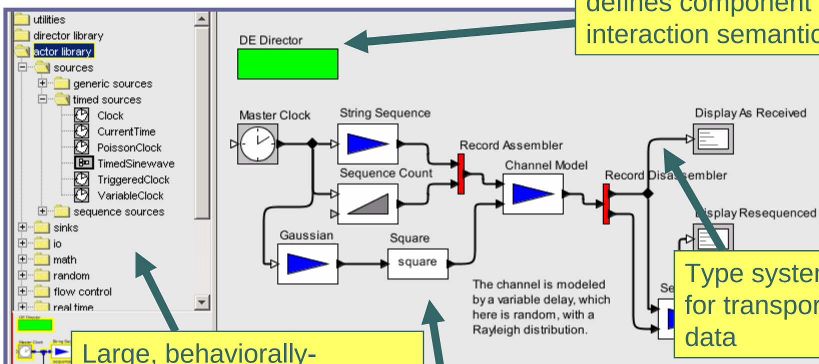
Ptolemy II is free software, open-source software

http://ptolemy.eecs.berkeley.edu

Ptolemy II Framework Supports Diverse Experiments with Models of Computation

Concurrency management supporting dynamic model structure.

Director from a library defines component interaction semantics



Type system for transported data

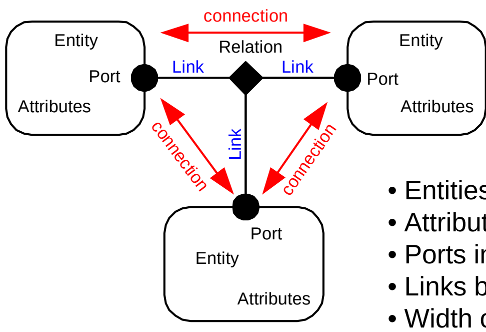
Large, behaviorally-polymorphic component library.

Visual editor supporting an abstract syntax

Separable Tool Architecture

- Abstract Syntax
- Concrete Syntax
- Abstract Semantics
- Concrete Semantics

The Basic Abstract Syntax for Composition

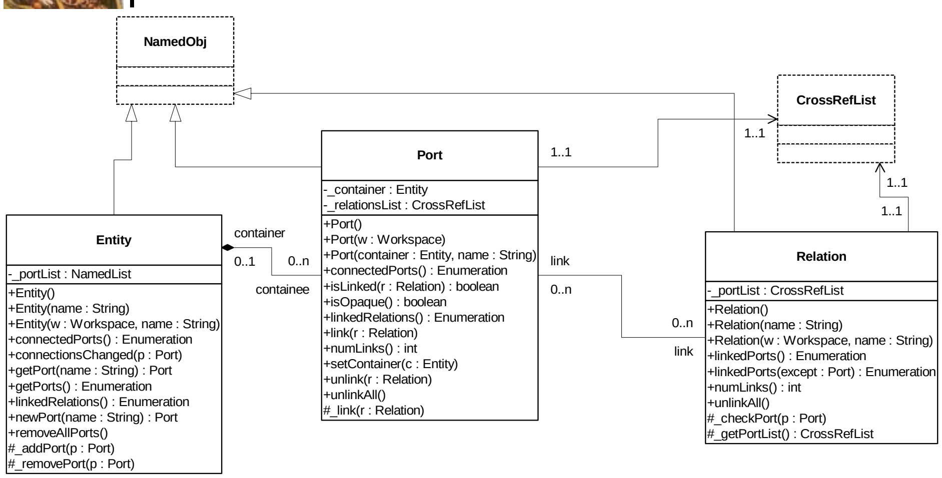


- Entities
- Attributes on entities (parameters)
- Ports in entities
- Links between ports
- Width on links (channels)
- Hierarchy

Concrete syntaxes:

- XML
- Visual pictures
- Actor languages (Cal, StreamIT, …)

Meta Model: Kernel Classes Supporting the Abstract Syntax



These get subclassed for specific purposes.

Separable Tool Archictecture

- Abstract Syntax
- Concrete Syntax
- Abstract Semantics
- Concrete Semantics

MoML
XML Schema for this Abstract Syntax

Ptolemy II designs are represented in XML:

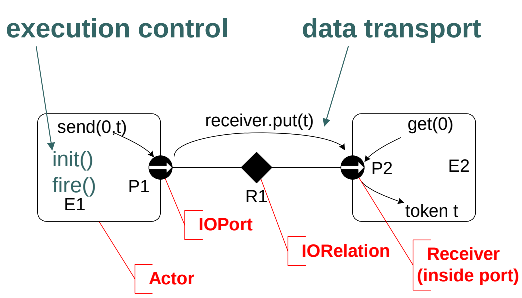
```
...
<entity name="FFT" class="ptolemy.domains.sdf.lib.FFT">
    <property name="order" class="ptolemy.data.expr.Parameter" value="order">
    </property>
    <port name="input" class="ptolemy.domains.sdf.kernel.SDFIOPort">
       ...
    </port>
    ...
</entity>
...
<link port="FFT.input" relation="relation"/>
<link port="AbsoluteValue2.output" relation="relation"/>
...
```

Separable Tool Archictecture

- Abstract Syntax
- Concrete Syntax
- Abstract Semantics
- Concrete Semantics

Abstract Semantics (Informally)

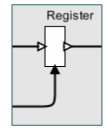
of *Actor-Oriented* Models of Computation



Actor-Oriented Models of Computation that we have implemented:

- dataflow (several variants)
- process networks
- distributed process networks
- Click (push/pull)
- continuous-time
- CSP (rendezvous)
- discrete events
- distributed discrete events
- synchronous/reactive
- time-driven (several variants)
- …

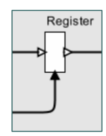
How Does This Work? Execution of Ptolemy II Actors



Flow of control:

- Initialization
- Execution
- Finalization

How Does This Work? Execution of Ptolemy II Actors

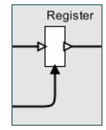


Flow of control:

- Initialization
- Execution
- Finalization

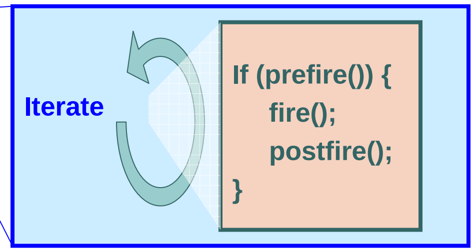
E.g., in DE: Post tags on the event queue corresponding to any initial events the actor wants to produce.

How Does This Work? Execution of Ptolemy II Actors



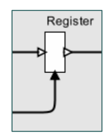
Flow of control:

- Initialization
- Execution
- Finalization



Only the postfire() method can change the state of the actor.

How Does This Work? Execution of Ptolemy II Actors



Flow of control:

- Initialization
- Execution
- Finalization

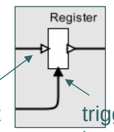
Definition of the Register Actor (Sketch)

```
class Register extends TypedAtomicActor {
  private Object state;
  boolean prefire() {
    if (trigger is known) { return true; }
  }
  void fire() {
    if (trigger is present) {
      send state to output;
    } else {
      assert output is absent;
    }
  }
  void postfire() {
    if (trigger is present) {
      state = value read from data input;
    }
  }
```

Can the actor fire? (prefire)

React to trigger input. (fire)

Read the data input and update the state. (postfire)



data input port

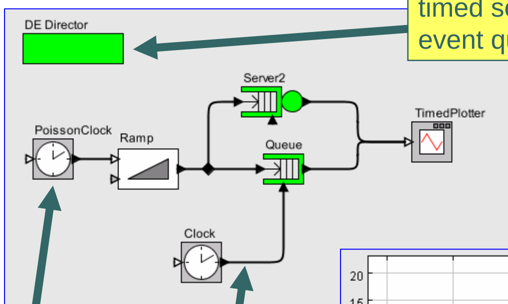
trigger input port

Separable Tool Archictecture

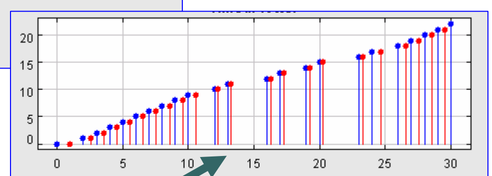
- Abstract Syntax
- Concrete Syntax
- Abstract Semantics
- Concrete Semantics

Concrete Semantics Example 1: Discrete Event (DE) Model of Computation (MoC)

DE Director implements timed semantics using an event queue



In DE, actors send time-stamped events to one another, and events are processed in chronological order.



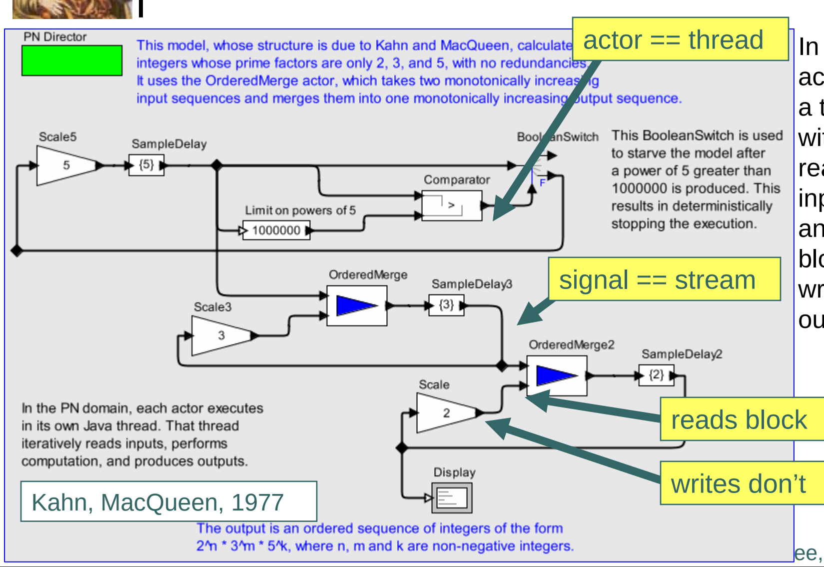
Event source

Signal

Time line

put() method inserts a token into the event queue.

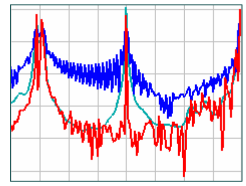
Example 2: Kahn Process Networks (PN) Model of Computation (MoC)

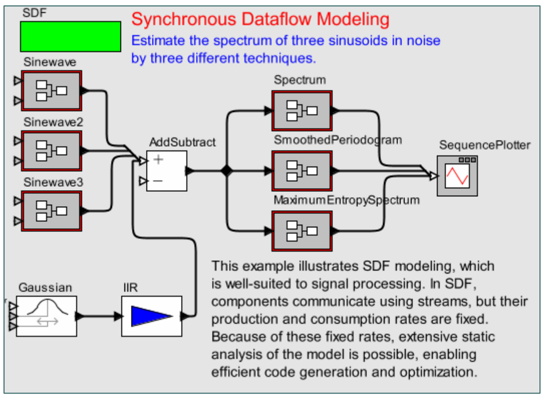


In PN, every actor runs in a thread, with blocking reads of input ports and non-blocking writes to outputs.

Example 3: Synchronous Dataflow (SDF)

In SDF, actors “fire,” and in each firing, consume a fixed number of tokens from the input streams, and produce a fixed number of tokens on the output streams.

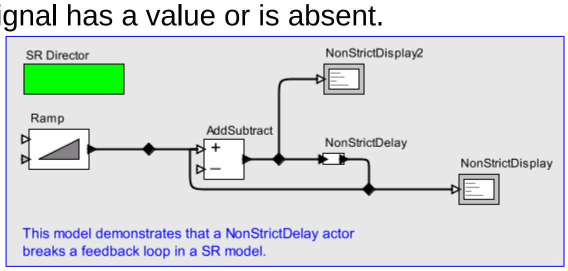


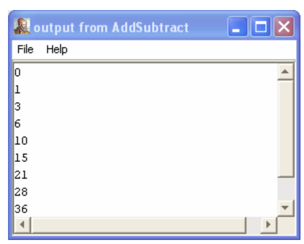


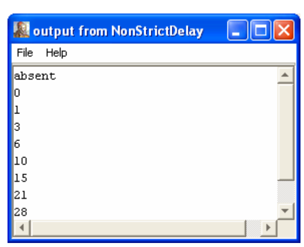
SDF is a special case of PN where deadlock and boundedness are decidable. It is well suited to static scheduling and code generation. It can also be automatically parallelized.

Example 4: Synchronous/Reactive (SR)

At each tick of a global “clock,” every signal has a value or is absent.



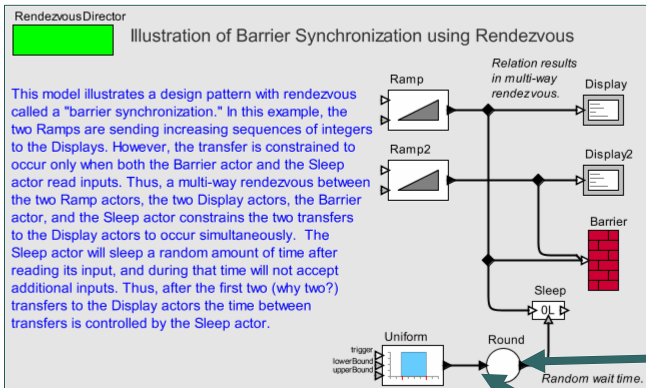




Like SDF, SR is decidable and suitable for code generation. It is harder to parallelize than SDF, however.

SR languages: Esterel, SyncCharts, Lustre, SCADE, Signal.

Example 5: Rendezvous



In Rendezvous, every actor runs in a thread, with blocking reads of input ports and blocking writes to outputs. Every communication is a (possibly multi-way) rendezvous.

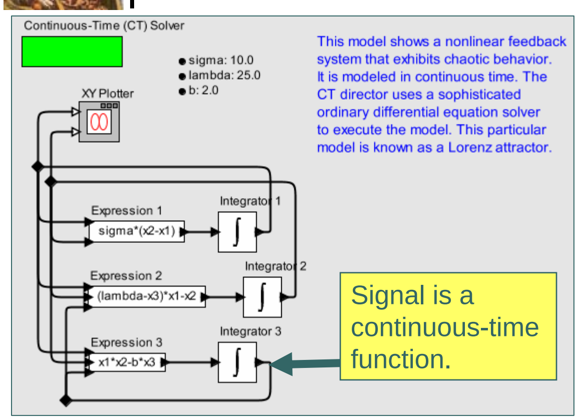
writes block

CSP (Hoare), SCCS (Milner), Reo (Arbab)

actor == thread

reads block

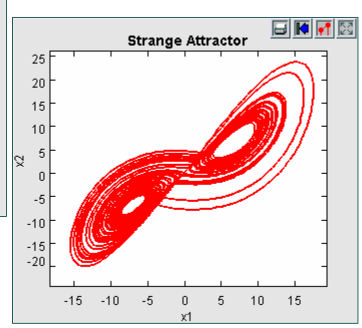
Example 6: Continuous Time (CT)



Signal is a continuous-time function.

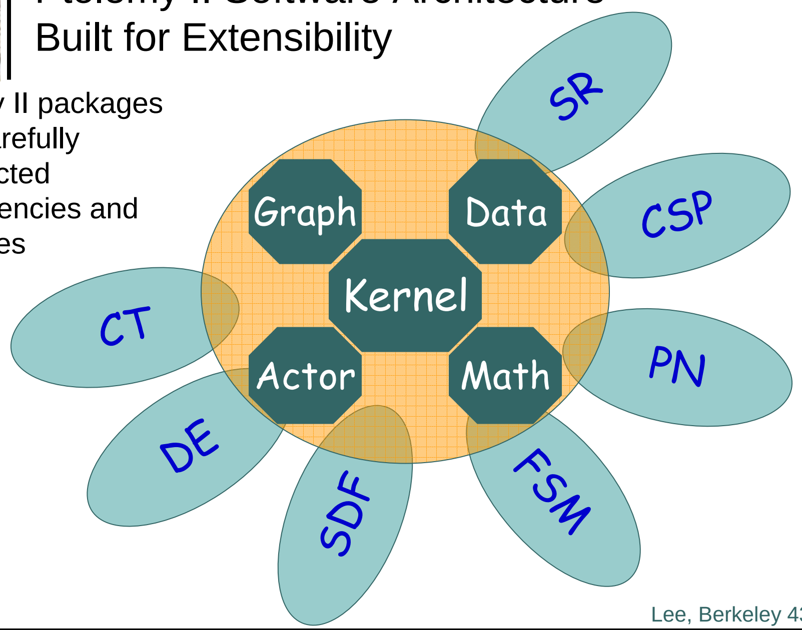
Director includes an ODE solver.

In CT, actors operate on continuous-time and/or discrete-event signals. An ODE solver governs the execution.



Ptolemy II Software Architecture Built for Extensibility

Ptolemy II packages have carefully constructed dependencies and interfaces



Lee, Berkeley 43

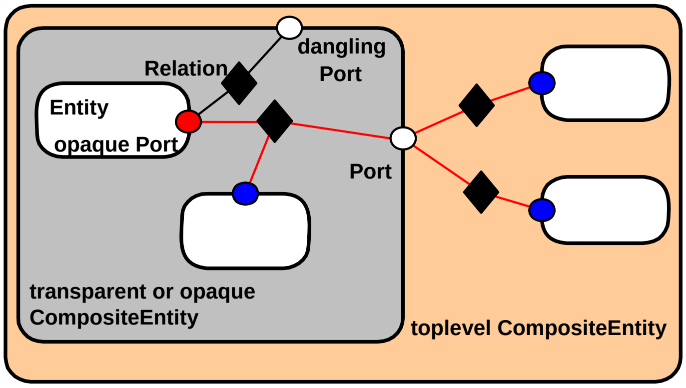
Models of Computation Implemented in Ptolemy II

- CI – Push/pull component interaction
- Click – Push/pull with method invocation
- CSP – concurrent threads with rendezvous
- Continuous – continuous-time modeling with fixed-point semantics
- CT – continuous-time modeling
- DDF – Dynamic dataflow
- DE – discrete-event systems
- DDE – distributed discrete events
- DPN – distributed process networks
- FSM – finite state machines
- DT – discrete time (cycle driven)
- Giotto – synchronous periodic
- GR – 3-D graphics
- PN – process networks
- Rendezvous – extension of CSP
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

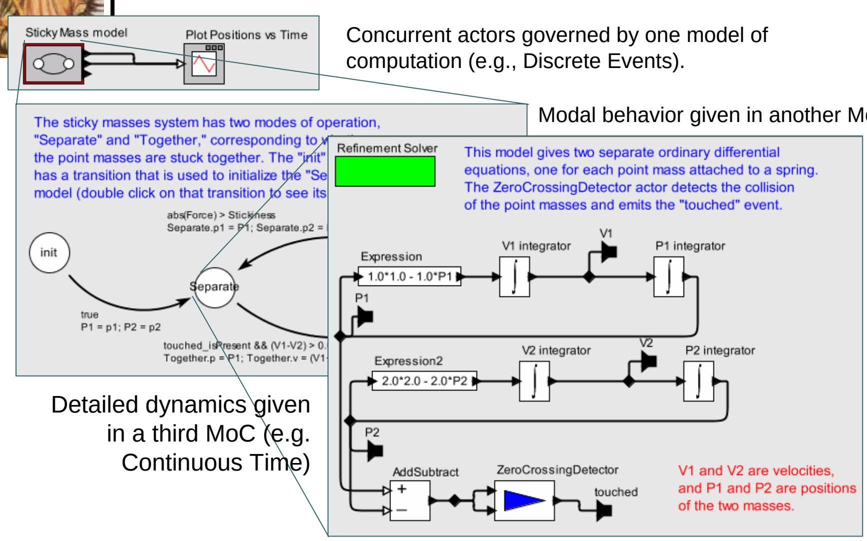
Most of these are actor oriented.

Lee, Berkeley 44

Scalability 101: Hierarchy - Composite Components



Ptolemy II Hierarchy Supports Heterogeneity



Concurrent actors governed by one model of computation (e.g., Discrete Events).

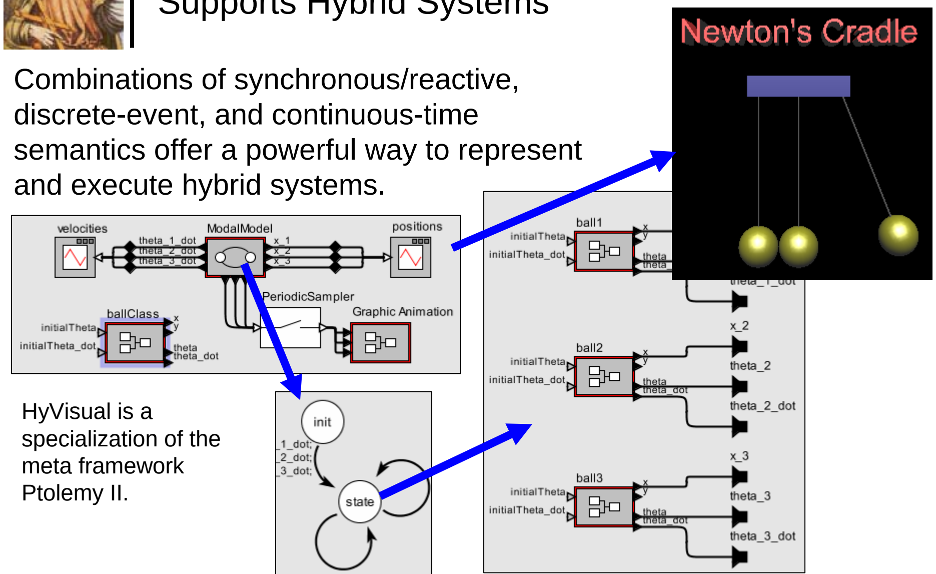
Modal behavior given in another MoC.

Detailed dynamics given in a third MoC (e.g. Continuous Time)

This requires a composable abstract semantics.

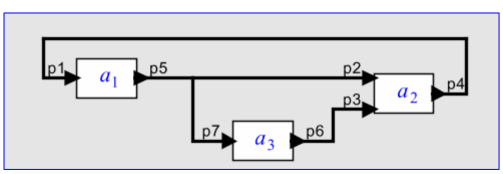
Hierarchical Heterogeneity (HH) Supports Hybrid Systems

Combinations of synchronous/reactive, discrete-event, and continuous-time semantics offer a powerful way to represent and execute hybrid systems.



HyVisual is a specialization of the meta framework Ptolemy II.

In All Cases: Composition Semantics



Each actor is a function:

$$f : (T \to B^*)^{\,m} \to (T \to B^*)^{\,n}$$

Composition in three forms:

- Cascade connections
- Parallel connections
- Feedback connections

All three are function composition.

The nontrivial part of this is feedback, but we know how to handle that.

The concurrency model is called the "model of computation" (MoC).

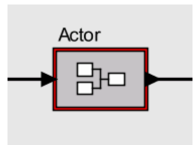
The model of computation determines the formal properties of the set T:

Useful MoCs:
- *Process Networks*
- *Synchronous/Reactive*
- *Time-Triggered*
- *Discrete Events*
- *Dataflow*
- *Rendezvous*
- *Continuous Time*
- *…*

Semantics Example: DE

signal in

F



signal out

A signal is a partial function:

$$F : \Re \times N \to T$$

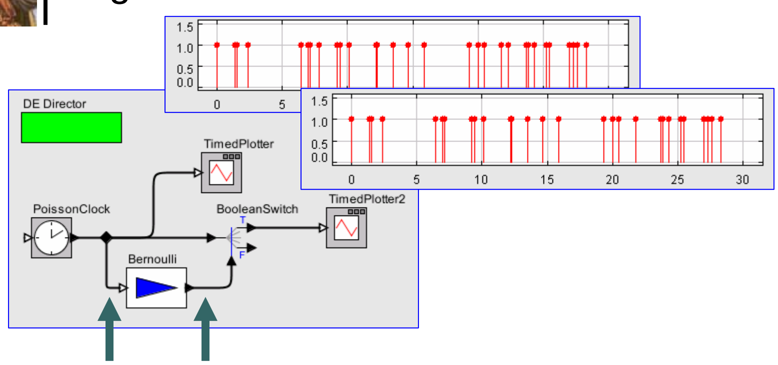
Data type (set of values)

Real numbers (approximated by doubles)

Natural numbers (allowing for simultaneous events in a signal)

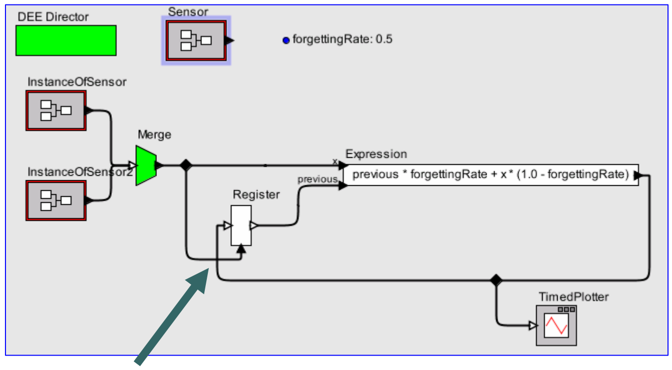
Note: A signal is not a single event but all the events that flow on a path.

Semantics Clears Up Subtleties: E.g. Simultaneous Events



By default, an actor produces events with the same time as the input event. But in this example, we expect (and need) for the BooleanSwitch to “see” the output of the Bernoulli in the same “firing” where it sees the event from the PoissonClock. Events with identical time stamps are also ordered, and reactions to such events follow data precedence order.

Semantics Clears Up Subtleties: E.g. Feedback



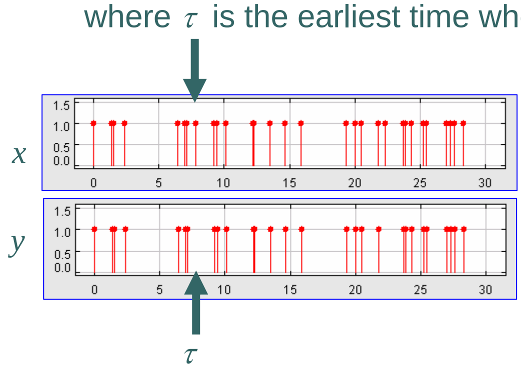
Data precedence analysis has to take into account the non-strictness of this actor (that an output can be produced despite the lack of an input).

Discrete-Event Semantics

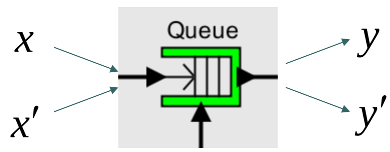
Cantor metric:

$$d(x, y) = 1/2^{\tau}$$

where τ is the earliest time where x and y differ.



Causality



Causal:

$$d(y, y') \le d(x, x')$$

Strictly causal:

$$d(y, y') < d(x, x')$$

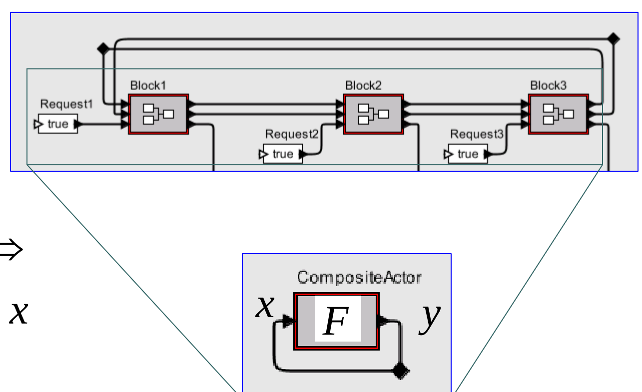
Delta causal:

$$\exists \delta < 1,$$
$$d(y, y') \le \delta\, d(x, x')$$

A delta-causal component is a “contraction map.”

Semantics of Composition

If the components are deterministic, the composition is deterministic.

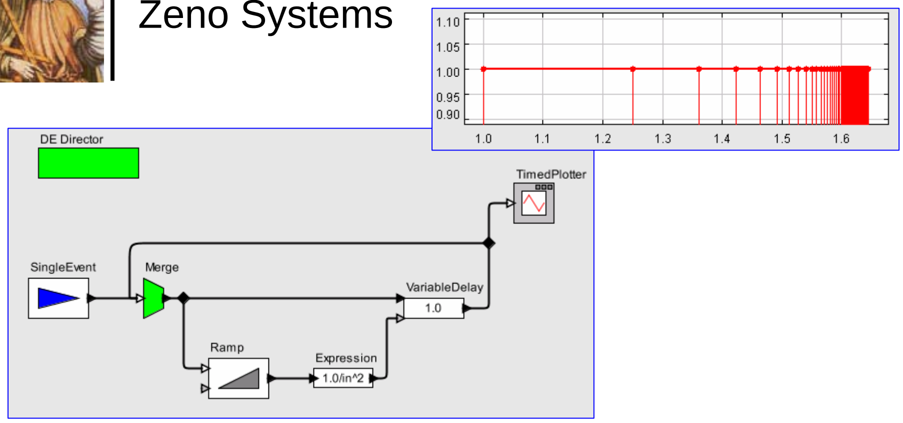


$$x = y \Rightarrow$$
$$F(x) = x$$

Banach fixed point theorem:

- Contraction map has a unique fixed point
- Execution procedure for finding that fixed point
- Successive approximations to the fixed point

Zeno Systems



Theorem: If every directed cycle contains a delta-causal component, then the system is non-Zeno.

Current Research in the Ptolemy Project

- **Precision-timed (PRET) machines**: This effort reintroduces timing into the core abstractions of computing, beginning with instruction set architectures, using configurable hardware as an experimental platform.
- **Real-time software**: Models of computation with time and concurrency, metaprogramming techniques, code generation and optimization, domain-specific languages, schedulability analysis, programming of sensor networks.
- **Distributed computing**: Models of computation based on distributed discrete events, backtracking techniques, lifecycle management, unreliable networks, modeling of sensor networks.
- **Understandable concurrency**: This effort focuses on models of concurrency in software that are more understandable and analyzable than the prevailing abstractions based on threads.
- **Systems of systems**: This effort focuses on modeling and design of large scale systems, those that include networking, database, grid computing, and information subsystems. See for example the Kepler project, which targets scientific workflows.
- **Abstract semantics**: Domain polymorphism, behavioral type systems, meta-modeling of semantics, comparative models of computation.
- **Hybrid systems**: Blended continuous and discrete dynamics, models of time, operational semantics, language design.

Install it!

- Latest release:
  - http://ptolemy.org
  - Follow Ptolemy II, downloads
- CVS tree:
  - http://chess.eecs.berkeley.edu/ptexternal/