

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.

Lee, Berkeley 2



Concurrent Composition of Subsystems, In Mainstream SW Engineering in 2007

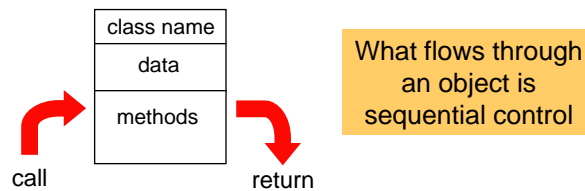
- Component technologies
 - Objects in C++, C#, or Java
 - Wrappers as service definitions
- Concurrency
 - Threads (shared memory, semaphores, mutexes, ...)
 - Message Passing (synchronous or not, buffered, ...)
- Distributed computing
 - Distributed objects wrapped in web services, Soap, CORBA, DCOM,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

Lee, Berkeley 3

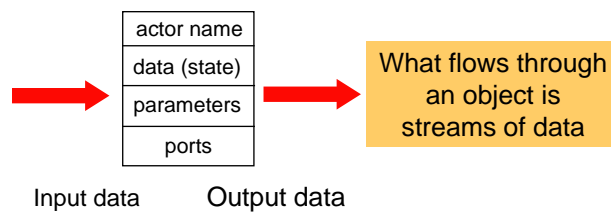


Our Approach: *Actor-Oriented Models of Computation*

Traditional component interactions:



Actor oriented:



Lee, Berkeley 4



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, Cl, GR, etc.

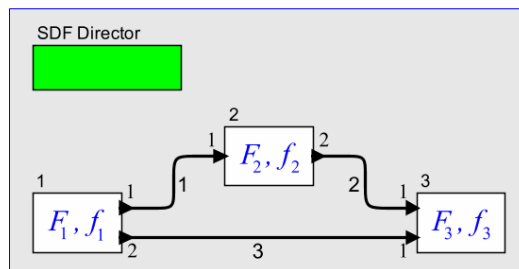
Each of these served us, first-and-foremost, as a laboratory for investigating design.

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

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

Actor 1

Connector 1

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

firing vector

balance equations

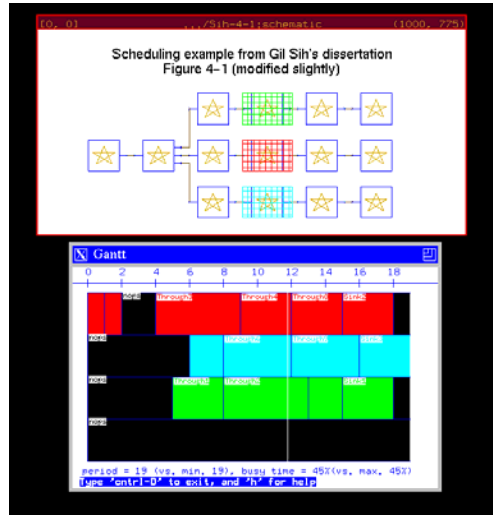
$$\Gamma q = \vec{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Lee, Berkeley 6



Gabriel and Ptolemy Classic Leveraged SDF to Generate Parallel Code

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



```
codeblock(Std) {
: initialize address registers for coef and
delayLineove #saddr(coef)+$val(coefLen)-1,r3
: insert here
move #ref(delayLineStart),r5
: delayLine
move #swal(stepSize),x1
move #ref(erase),x0
mqr x0,x1,a
move a,x0
move x:(r3),b u:(r5)+,y0
}

codeblock(loop) {
do #swal(loopVal), $label(endlloop)
macr x0,y0,b
move b,x:(r3)-
move x:(r3),b u:(r5)+,y0
$label(endlloop)
}

codeblock(noloop) {
macr x0,y0,b
move b,x:(r3)-
move x:(r3),b u:(r5)+,y0
}
```

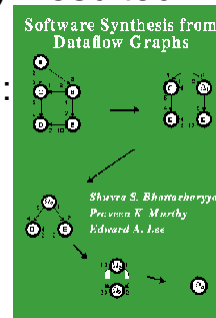
It is an interesting (and rich) research problem to minimize interlocks in complex multirate applications.

Lee, Berkeley 7



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).

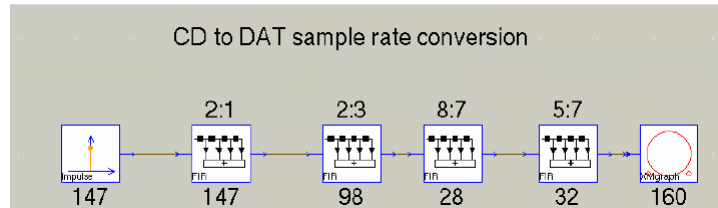


See S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*, Kluwer Academic Press, 1996, for a summary of the single processor optimization problems.

Lee, Berkeley 8



Minimum Buffer Schedule



```

ABABCABCABABCABCDEAFFFFFABCABCABABCDE
AFFFFFBCABABCABCABABCDEAFFFFFBCABABCABC
DEAFFFFFBCABABCABCABABCDEAFFFFFBCABABCABC
BABCDEAFFFFFBCABABCABCABABCDEAFFFFFBCA
FFFFFABCABCABCDEAFFFFFABCABCABABCABCDEAF
FFFFFABCABCABCABABCDEAFFFFFBCABABCABCABABC
DEAFFFFFBCABABCABCDEAFFFFFBCABABCABCABABC
BCDEAFFFFFABCABCABABCDEAFFFFFBCAFFFFFBA
ABCABCABABCDEAFFFFFBCABABCABCDEAFFFFFBA
BCABCABABCABCDEAFFFFFABCABCABABCDEAFFFF
FFBCABABCABCABABCDEAFFFFFBCABABCABCDEAF
FFFFFABCABCABABCABCDEAFFFFFBCAFFFFFBCABC
ABABCDEAFFFFFBCABABCABCABABCDEAFFFFFBCA
BABCABCDEAFFFFFABCABCABABCABCDEAFFFFF
ABCABCABABCDEAFFFFFBCABABCABCABABCDEAF
FFFFFBCABABCABCDEAFFFFFBCAFFFFF
  
```

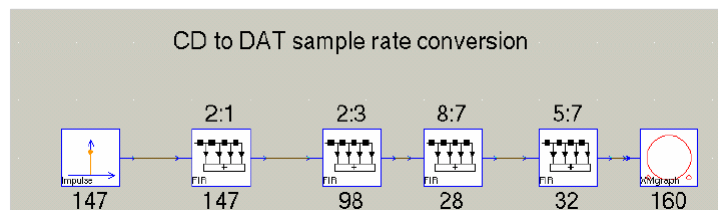
Source: Shuvra Bhattacharyya

Lee, Berkeley 9



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

Lee, Berkeley 10



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

The screenshot displays the Thor Analyzer interface. On the left, there's a block diagram of a logic analyzer with 'Add' and 'Mux' blocks. The top right shows a timing diagram for signals like CL001.clk, DSP6000_01, DSP6000_02, and DSP6000_03. The bottom left is a terminal window for the DSP6000 simulator, showing commands like 'command: cmd', 'RPC connect', and 'load code!'. The bottom right shows a schematic diagram of two DSP6000 processors connected to a central logic analyzer block.

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).

Lee, Berkeley 11



Example of Model-Based Design: ADPCM Speech Coding

The screenshot shows the VEM (Virtual Embedded Model) software interface for ADPCM speech coding. It features several windows: 'Adaptive DPCM Speech Coding' showing a block diagram with TX and RX blocks; 'VEM call' windows showing signal flow and processing; 'Universe Parameter Control' with a 'Quantization Range' set to 0.0 to 0.3089; and a 'Running' window displaying a table of registers and memory values.

Quit	Load	Process	Breakpoints	Registers	Memory
x1=	9f1d34	nd=	9f1d34	r7=	0000 n7= 0000 n7= ffff
y1=	0d031f	pb=	7ffffe	r5=	0000 n5= 0040 n5= ffff
nd=	ff b1=	0d0056	bd=	c28b30	r5= 0020 n5= 0001 n5= ffff
bd=	00 b1=	000000	bd=	000000	r4= 001c n4= 0001 n4= ffff
pc=	7777	ac=	7777	rac=	00 r3= 0022 n3= 0000 n3= ffff
lab=	003b	ic=	0200	rg=	01 r2= c000 n2= 0001 n2= ffff
					r1= c003 n1= 0000 n1= ffff

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

Lee, Berkeley 12



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.

Control panel for synthDisplayFFT0:

- GO <Return> PAUSE <Space> STOP <Escape>
- Number of Iterations: -1
- FM_Index: 0.7600
- Volume: 0.6300
- QUIT

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)

An Adaptive Array Processor with a 4 Element Uniform Circular Array suppresses three Cochannel Interferers

higher-order components

streams

hierarchical components

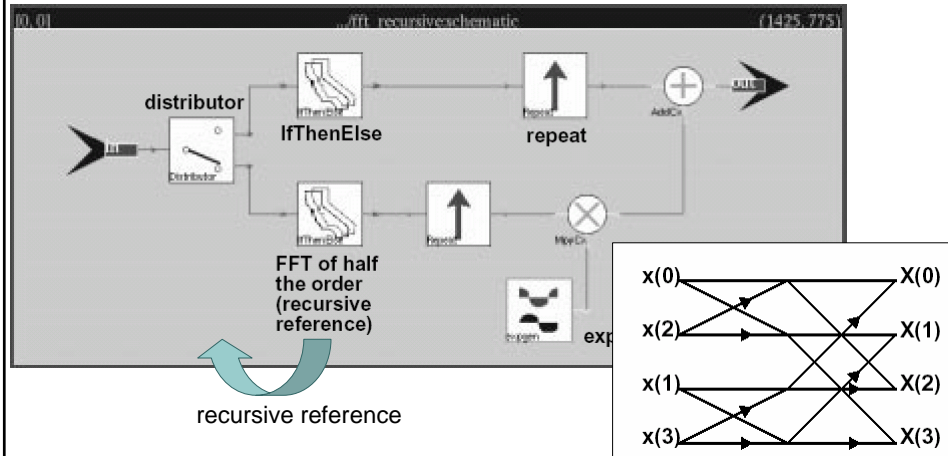
Ptolemy application developed by Uwe Trautwein, Technical University of Ilmenau, Germany

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.



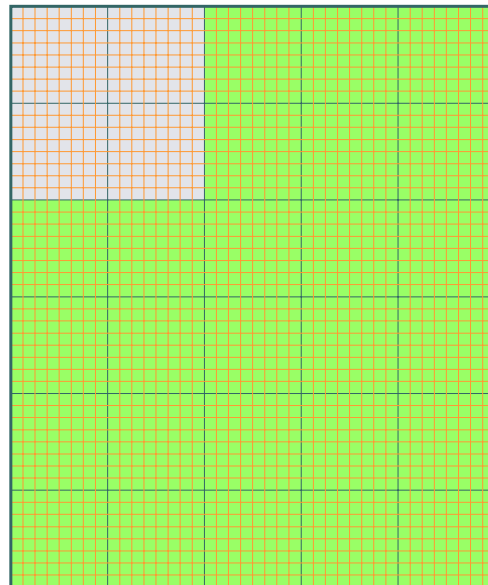
Extension of SDF: Multidimensional SSDF (1993)

- 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.



Lee, Berkeley 16



More interesting Example

Two dimensional FFT constructed out of one-dimensional actors.

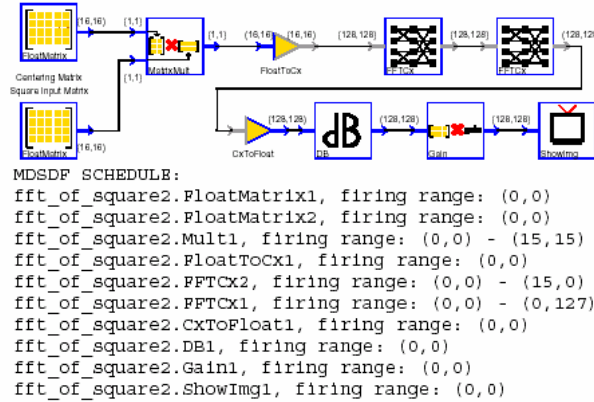
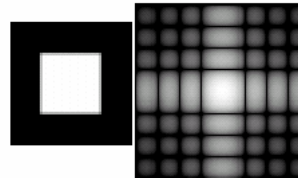
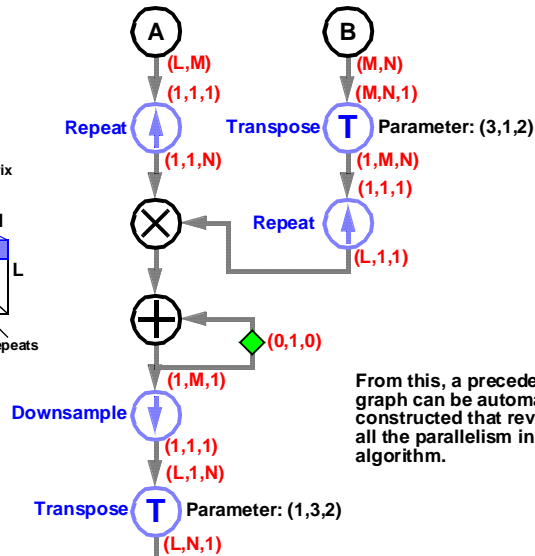
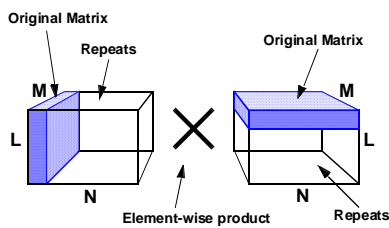


Figure 6. Screen dump of 2D-FFT system, the associated schedule, and outputs.



MDSSDF Structure Exposes Fine-Grain Data Parallelism



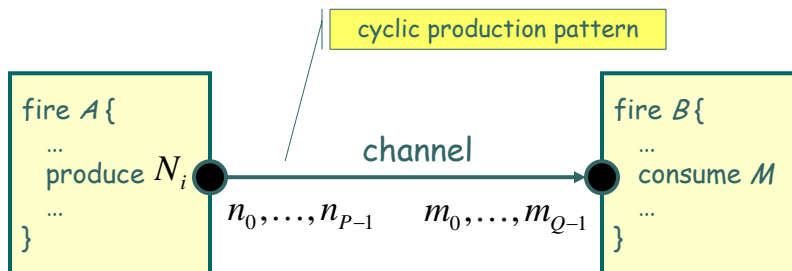
However, such programs are extremely hard to write (and to read).



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

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

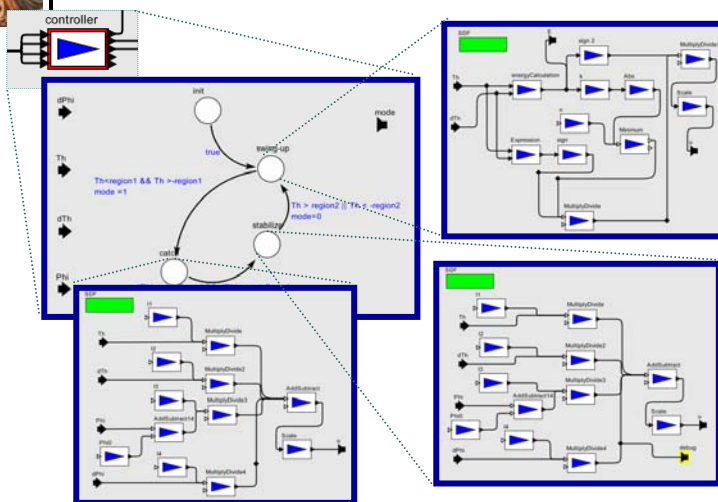
$$q_A \sum_{i=0}^{R-1} n_{i \bmod P} = q_B \sum_{i=0}^{R-1} m_{i \bmod Q}; R = \text{lcm}(P, Q)$$



Lee, Berkeley 19



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)

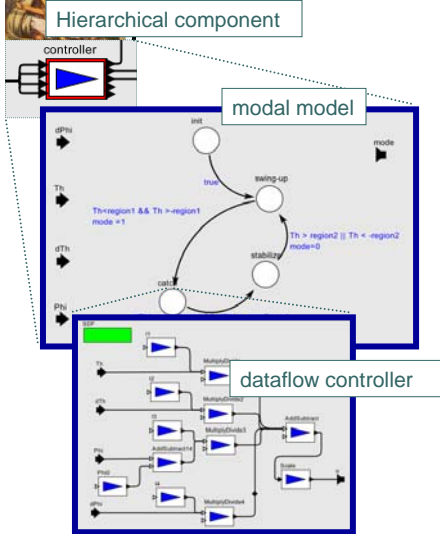
Related to "parameterized dataflow" of Bhattacharya and Bhattacharyya (2001).

- 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

Lee, Berkeley 21



Ptolemy II



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>

example Ptolemy II model: hybrid control system

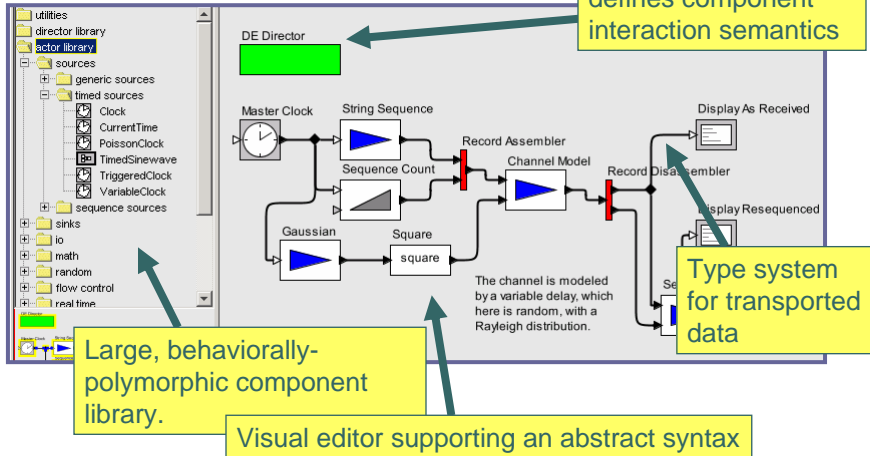
Lee, Berkeley 22



Ptolemy II Framework Supports Diverse Experiments with Models of Computation

Concurrency management supporting dynamic model structure.

Director from a library defines component interaction semantics



Lee, Berkeley 23



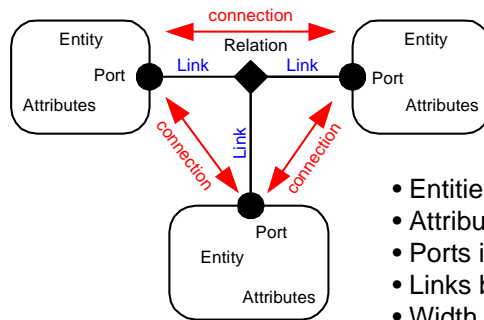
Separable Tool Architecture

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

Lee, Berkeley 24



The Basic Abstract Syntax for Composition



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

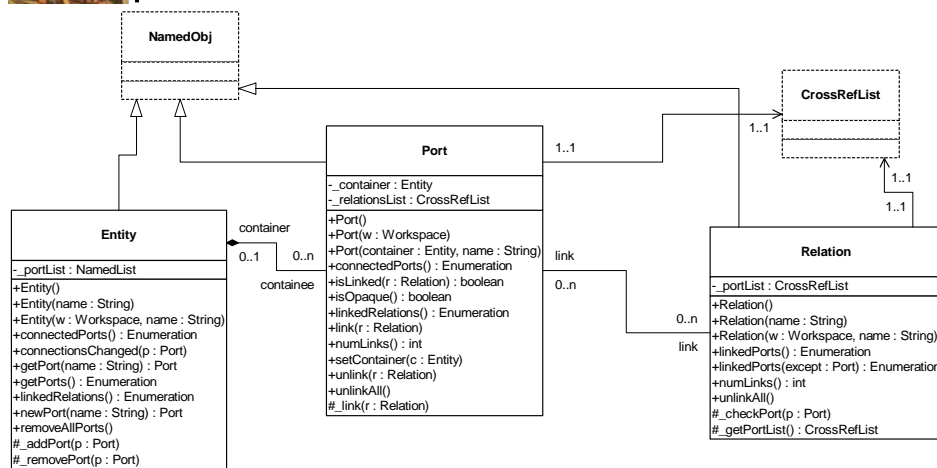
Concrete syntaxes:

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

Lee, Berkeley 25



Meta Model: Kernel Classes Supporting the Abstract Syntax



These get subclassed for specific purposes.

Lee, Berkeley 26



Separable Tool Architecture

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

Lee, Berkeley 27



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"/>  
...
```

Lee, Berkeley 28



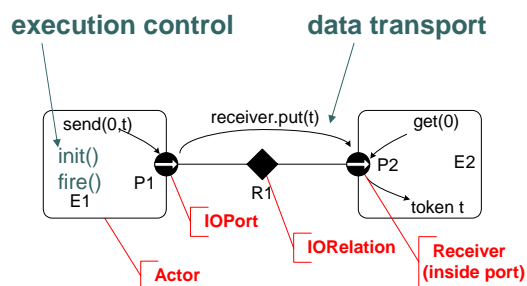
Separable Tool Architecture

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

Lee, Berkeley 29



Abstract Semantics (Informally) of Actor-Oriented Models of Computation



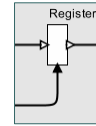
Actor-Oriented Models of Computation that we have implemented:

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

Lee, Berkeley 30



How Does This Work? Execution of Ptolemy II Actors



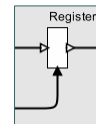
Flow of control:

- Initialization
- Execution
- Finalization

Lee, Berkeley 31



How Does This Work? Execution of Ptolemy II Actors



Flow of control:

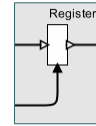
- Initialization
- Execution
- Finalization

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

Lee, Berkeley 32

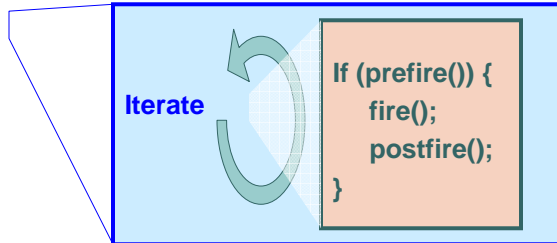


How Does This Work? Execution of Ptolemy II Actors



Flow of control:

- Initialization
- Execution
- Finalization

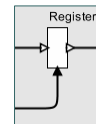


Only the postfire() method can change the state of the actor.

Lee, Berkeley 33



How Does This Work? Execution of Ptolemy II Actors



Flow of control:

- Initialization
- Execution
- Finalization

Lee, Berkeley 34



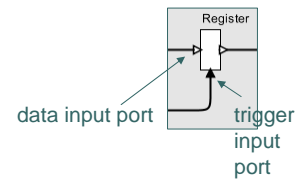
Definition of the Register Actor (Sketch)

```
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?

React to trigger input.

Read the data input and update the state.



Lee, Berkeley 35



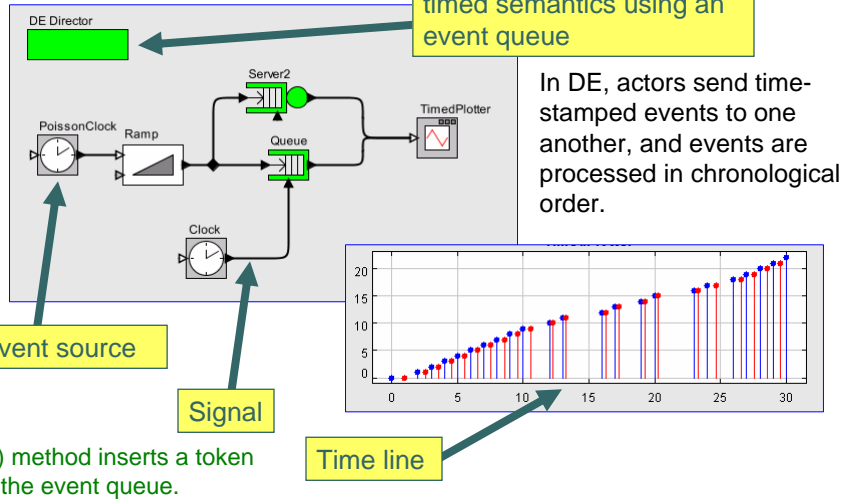
Separable Tool Architecture

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

Lee, Berkeley 36



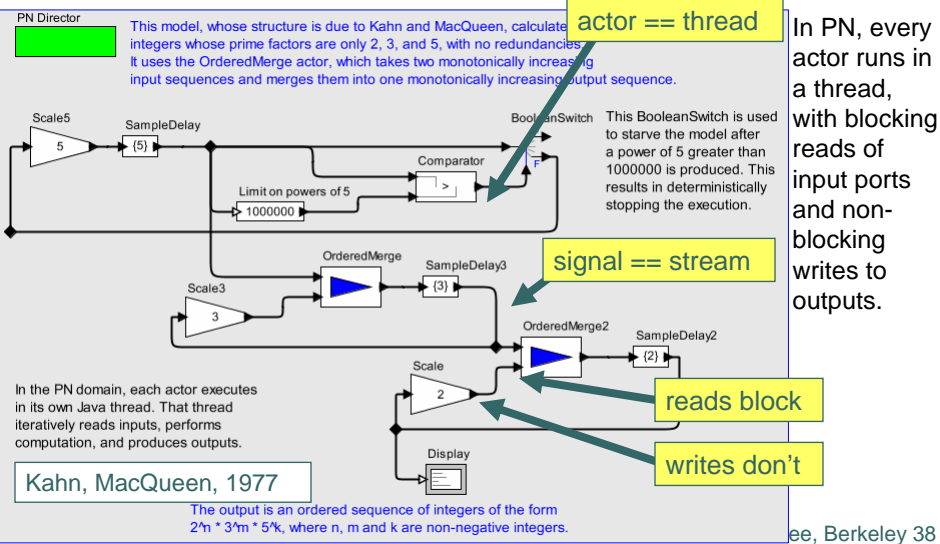
Concrete Semantics Example 1: Discrete Event (DE) Model of Computation (MoC)



Lee, Berkeley 37



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

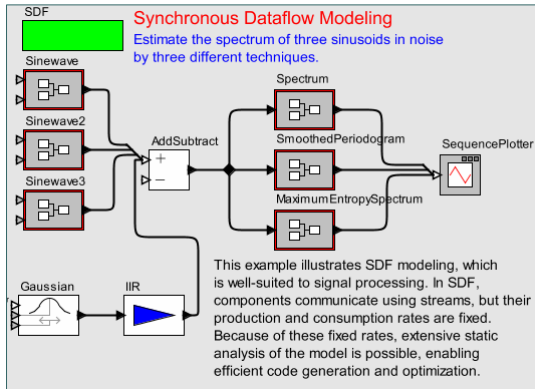
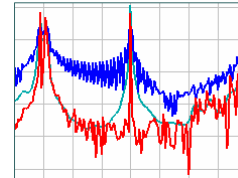


Lee, Berkeley 38



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.



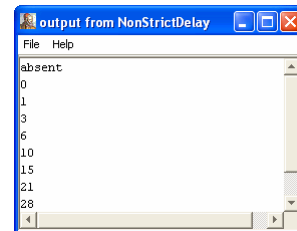
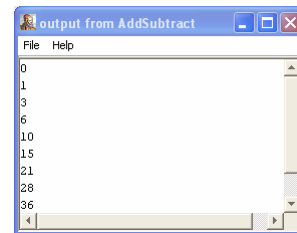
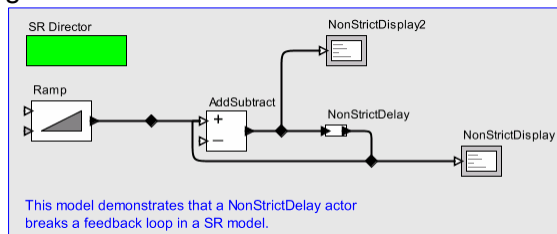
SDF is a special case of PN where deadlock and boundedness are decidable. It is well suited to static scheduling and code generation. It can also be automatically parallelized.

Lee, Berkeley 39



Example 4: Synchronous/Reactive (SR)

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



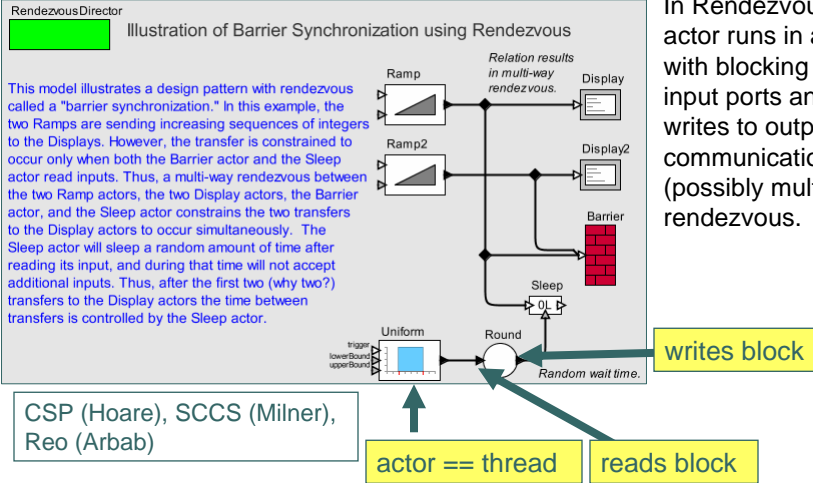
Like SDF, SR is decidable and suitable for code generation. It is harder to parallelize than SDF, however.

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

Lee, Berkeley 40



Example 5: Rendezvous

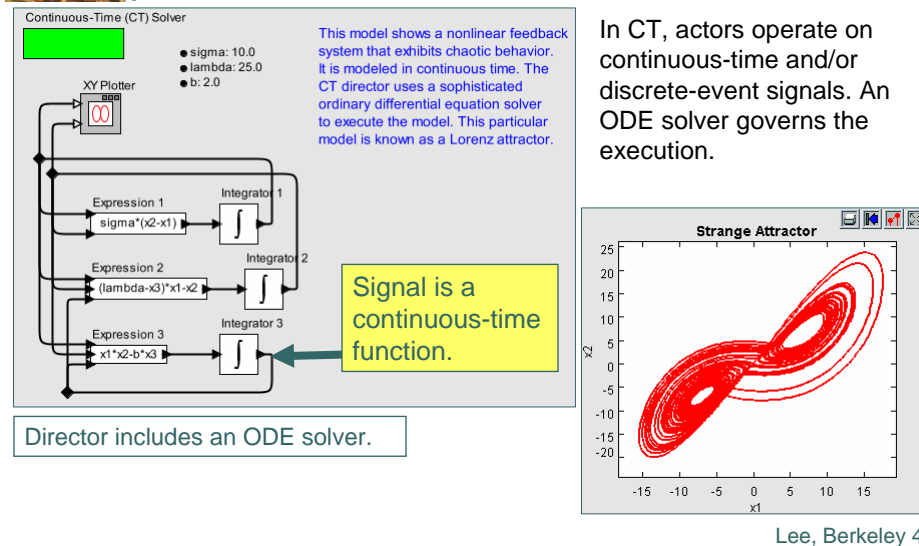


In Rendezvous, every actor runs in a thread, with blocking reads of input ports and blocking writes to outputs. Every communication is a (possibly multi-way) rendezvous.

Lee, Berkeley 41



Example 6: Continuous Time (CT)

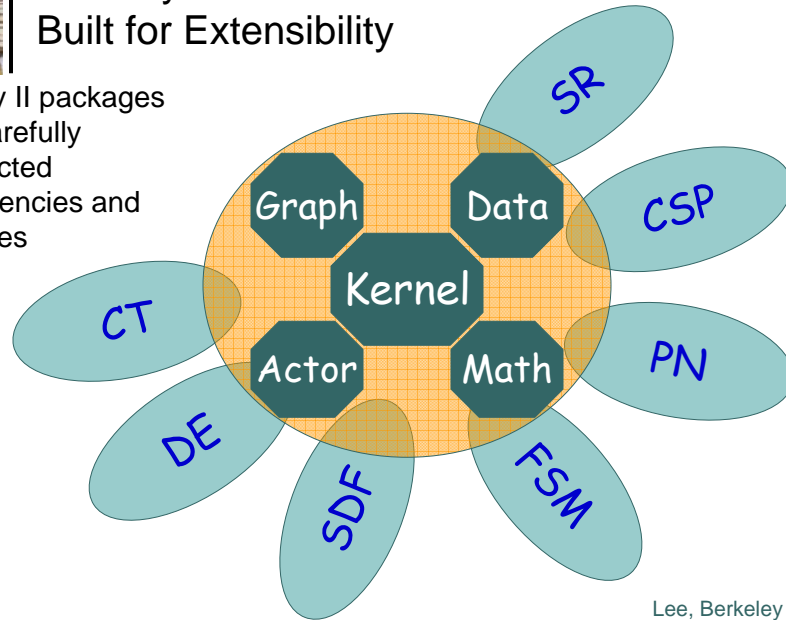


In CT, actors operate on continuous-time and/or discrete-event signals. An ODE solver governs the execution.



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

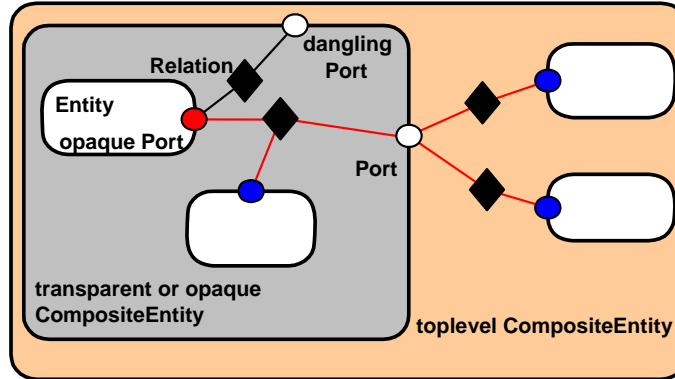
- CI – Push/pull component interaction
- Click – Push/pull with method invocation
- CSP – concurrent threads with rendezvous
- Continuous – continuous-time modeling with fixed-point semantics
- CT – continuous-time modeling
- DDF – Dynamic dataflow
- DE – discrete-event systems
- DDE – distributed discrete 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



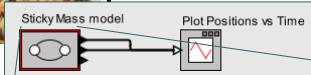
Scalability 101: Hierarchy - Composite Components



Lee, Berkeley 45



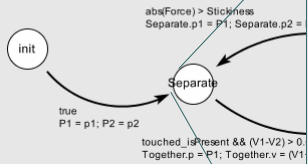
Ptolemy II Hierarchy Supports Heterogeneity



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

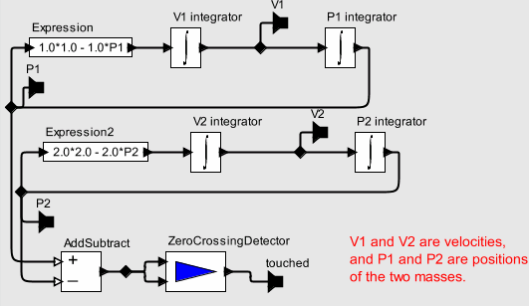
The sticky masses system has two modes of operation, "Separate" and "Together," corresponding to the point masses are stuck together. The "init" has a transition that is used to initialize the "Separate" model (double click on that transition to see its

Modal behavior given in another MoC.



Refinement Solver This model gives two separate ordinary differential equations, one for each point mass attached to a spring. The ZeroCrossingDetector actor detects the collision of the point masses and emits the "touched" event.

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



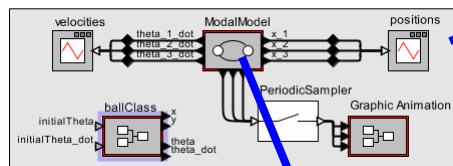
This requires a composable abstract semantics.

Lee, Berkeley 46

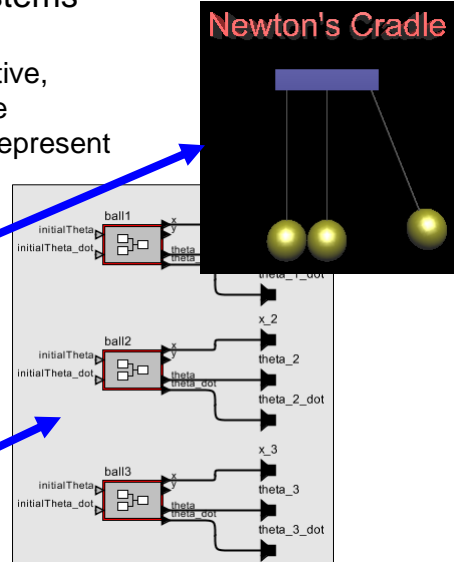
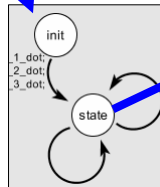


Hierarchical Heterogeneity (HH) Supports Hybrid Systems

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



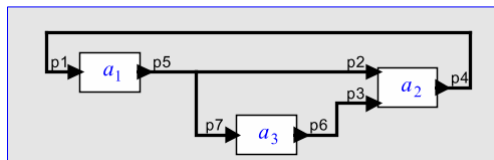
HyVisual is a specialization of the meta framework Ptolemy II.



Lee, Berkeley 47



In All Cases: Composition Semantics



Each actor is a function:

$$f: (T \rightarrow B^*)^m \rightarrow (T \rightarrow B^*)^n$$

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:

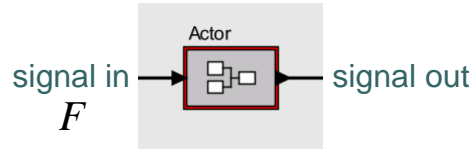
Useful MoCs:

- Process Networks
- Synchronous/Reactive
- Time-Triggered
- Discrete Events
- Dataflow
- Rendezvous
- Continuous Time
- ...

Lee, Berkeley 48



Semantics Example: DE



A signal is a partial function:

$$F : \mathcal{R} \times \mathcal{N} \rightarrow T$$

Real numbers
(approximated
by doubles)

Natural numbers (allowing
for simultaneous events in
a signal)

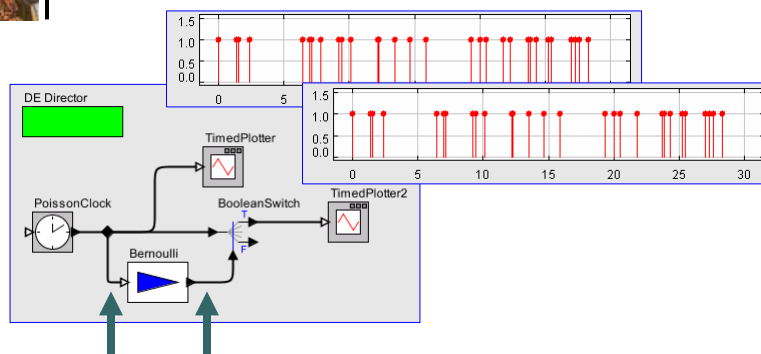
Data type
(set of
values)

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

Lee, Berkeley 49



Semantics Clears Up Subtleties: E.g. Simultaneous Events

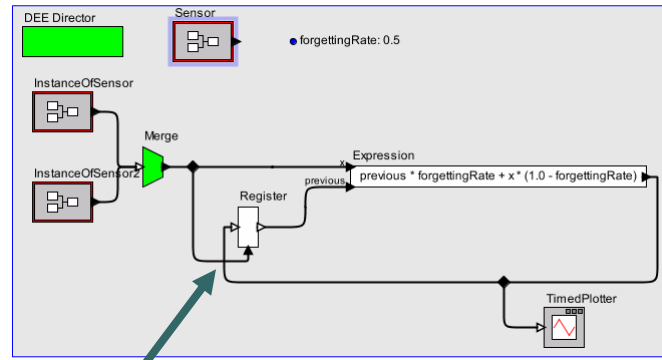


By default, an actor produces events with the same time as the input event. But in this example, we expect (and need) for the BooleanSwitch to “see” the output of the Bernoulli in the same “firing” where it sees the event from the PoissonClock. Events with identical time stamps are also ordered, and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

Lee, Berkeley 50



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).

Lee, Berkeley 51

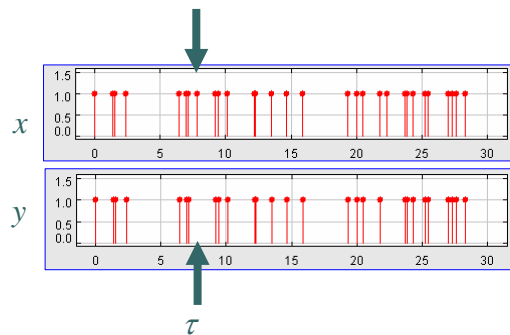


Discrete-Event Semantics

Cantor metric:

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

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



Lee, Berkeley 52



Causality

Causal:

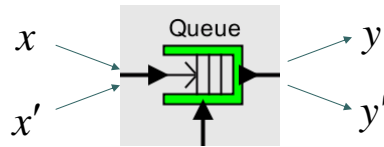
$$d(y, y') \leq d(x, x')$$

Strictly causal:

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

Delta causal:

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



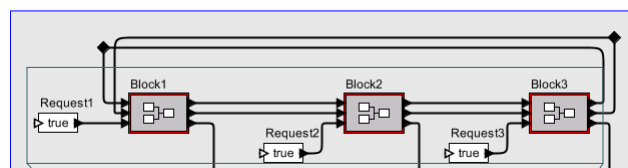
A delta-causal component is a “contraction map.”

Lee, Berkeley 53

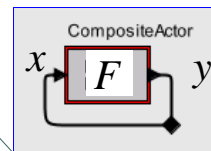


Semantics of Composition

If the components are deterministic, the composition is deterministic.



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



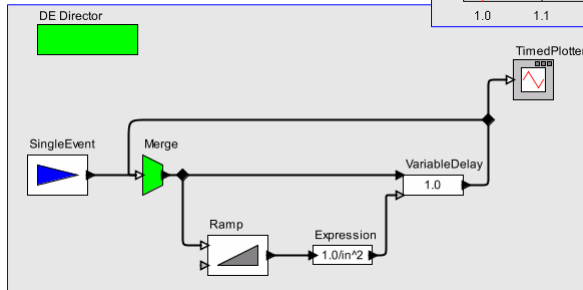
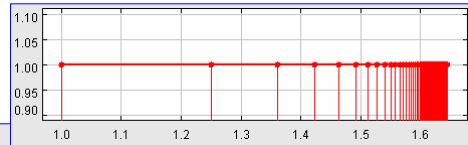
Banach fixed point theorem:

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

Lee, Berkeley 54



Zeno Systems



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

Lee, Berkeley 55



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.

Lee, Berkeley 56



Install it!

- Latest release:
 - <http://ptolemy.org>
 - Follow Ptolemy II, downloads

- CVS tree:
 - <http://chess.eecs.berkeley.edu/ptexternal/>

Lee, Berkeley 57