

## Modeling, Simulation, and Design of Concurrent Real-Time Embedded Systems Using Ptolemy

Edward A. Lee

*Robert S. Pepper Distinguished Professor  
EECS Department  
UC Berkeley*

**Ptutorial**

*EECS 249, Sept. 13, 2012*



## The Ptolemy Project

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



## The Ptolemy Project Demographics, 2012

### Sponsors:

- **Government**
  - *National Science Foundation*
  - *Army Research Lab*
  - *DARPA (MuSyC: Multiscale Systems Center)*
  - *Air Force Research Lab*
- **Industry**
  - *Bosch*
  - *National Instruments*
  - *SRC (MuSyC: Multiscale Systems Center)*
  - *Thales*
  - *Toyota*

### History:

*The project was started in 1990, though its mission and focus has evolved considerably. An open-source, extensible software framework (Ptolemy II) constitutes the principal experimental laboratory.*

### Staffing:

- 1 professor
- 9 graduate students
- 3 postdocs
- 2 research staff
- several visitors

Lee, Berkeley 3



## Contributors to Ptolemy II

### Principal Authors

- Christopher Brooks
- Dai Bui
- Chamberlain Fong
- John Davis, II
- Patricia Derler
- Thomas Huining Feng
- Mudit Goel
- Rowland Johnson
- Bilung Lee
- Edward Lee
- Ben Lickly
- Jie Liu
- Xiaojun Liu
- Lukito Muliadi
- Stephen Neuendorffer
- John Reekie
- Neil Smyth
- Jeff Tsay
- Yuhong Xiong
- Haiyang Zheng
- Gang Zhou

### Other Contributors

- Jim Armstrong
- Vincent Arnould
- Kyungmin Bae
- Philip Baldwin
- Chad Berkley
- Frederic Boulanger
- Raymond Cardillo
- Jannette Cardoso
- Adam Cataldo
- Christine Cavanessians
- Chris Chang
- Albert Chen
- Chihong Patrick Cheng
- Elaine Cheong
- Colin Cochran
- Brieuc Desoutter
- Pedro Domecq
- William Douglas
- Johan Eker
- Thomas Huining Feng
- Tobin Fricke
- Teale Fristoe
- Shanna-Shaye Forbes
- Hauke Fuhrmann
- Geroncio Galicia
- Ben Horowitz
- Heloise Hse
- Efrat Jaeger
- Jörn Janneck
- Zoltan Kemenczy
- Bart Kienhuis
- Christoph Meyer Kirsch
- Sanjeev Kohli
- Vinay Krishnan
- Robert Kroeger
- Daniel Lázaro Cuadrado
- David Lee
- Man-kit (Jackie) Leung
- Michael Leung
- John Li
- Isaac Liu
- Andrew Mihal
- Eleftherios Matsikoudis
- Aleksandar Necakov
- Mike Kofi Okyere
- Sarah Packman
- Shankar Rao
- Bert Rodiers
- Rakesh Reddy
- Adriana Ricchiuti
- Sonia Sachs
- Ismael M. Sarmiento
- Michael Shilman

- Sean Simmons
- Mandeep Singh
- Miro Spoenemann
- Peter N. Steinmetz
- Dick Stevens
- Mary Stewart
- Ned Stoffel
- Manda Sutijono
- Stavros Tripakis
- Neil Turner
- Guillaume Vibert
- Kees Vissers
- Brian K. Vogel
- Yuke Wang
- Xavier Warzee
- Scott Weber
- Paul Whitaker
- Winthrop Williams
- Ed Willink
- Michael Wirthlin
- Michael Wetter
- William Wu
- Xiaowen Xin
- Paul Yang
- James Yeh
- Nick Zamora
- Charlie Zhong

Lee, Berkeley 4



## References

- Ptolemy project home page:  
<http://ptolemy.org>
- Latest release:  
<http://ptolemy.org/ptolemyII/ptIIlatest/>
- Latest version in the SVN repository:  
<http://chess.eecs.berkeley.edu/ptexternal/>

Lee, Berkeley 5



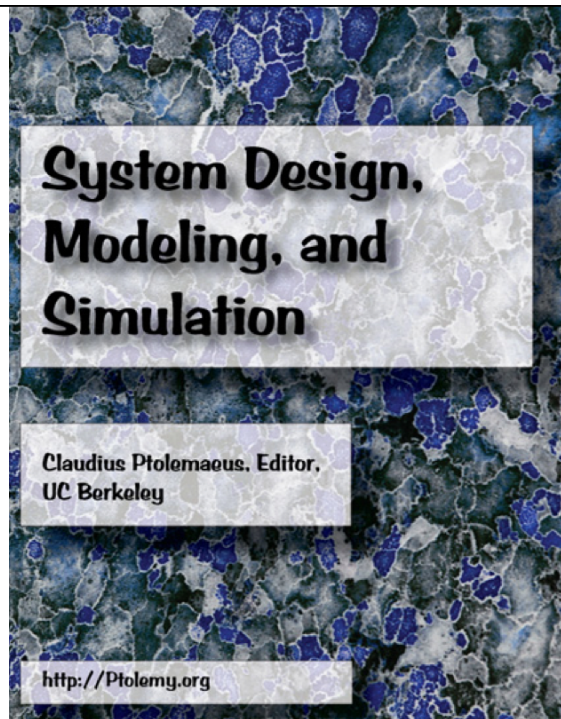
## Forthcoming Book

### Chapters

1. Heterogeneous Modeling
2. Building Graphical Models
3. Dataflow
4. Process Networks and Rendezvous
5. Synchronous/Reactive Models
6. Finite State Machines
7. Discrete Event Models
8. Modal Models
9. Continuous Time Models
10. Cyber-Physical Systems

### Appendices

- A. Expressions
- B. Signal Display
- C. The Type System
- D. Creating Web Pages



# Getting More Information: Documentation



## PTOLEMY II HETEROGENEOUS CONCURRENT MODELING AND DESIGN IN JAVA

Edited by:  
Christopher Hylton, Edward A. Lee, Xu Liu, Xiaojun  
Liu, Steve Neundorfer, Yukang Xiong, Hsiangping Zhang

### VOLUME 1: INTRODUCTION TO PTOLEMY II

Authors:  
Shawn S. Bhattacharya  
Elaine Chong  
John Davis, II  
Mehdi Dool  
Burt Kianian  
Christopher Hylton  
Edward A. Lee  
Xu Liu  
Xiaojun Liu  
Lukho Mikhali  
Steve Neundorfer  
John Reule  
Neil Smith  
Jeff Top  
Brian Vogel  
Whitney Williams  
Yukang Xiong  
Tong Zhao  
Hsiangping Zhang

Department of Electrical Engineering and Computer Sciences  
University of California at Berkeley  
<http://ptolemy.eecs.berkeley.edu>

Document Version 3.0  
In use with Ptolemy II 3.0  
June 8, 2005

Memorandum UCBERE-M0578A

Earlier versions:  
• UCBERE-M0523  
• UCBERE-M0549  
• UCBERE-M0512

This project is supported by the Defense Advanced Research Projects Agency (DARPA), the National Science Foundation, Chou the Center for Hybrid and Embedded Software Systems, the State of California MICRO program, and the following companies: Agilent, Amel, Cadence, Intel, Intelwave, National Semiconductor, Philips, and Wind River Systems.



Volume 1:  
User-Oriented



## PTOLEMY II HETEROGENEOUS CONCURRENT MODELING AND DESIGN IN JAVA

Edited by:  
Christopher Hylton, Edward A. Lee, Xu Liu, Xiaojun  
Liu, Steve Neundorfer, Yukang Xiong, Hsiangping Zhang

### VOLUME 2: PTOLEMY II SOFTWARE ARCHITECTURE

Authors:  
Shawn S. Bhattacharya  
Elaine Chong  
John Davis, II  
Mehdi Dool  
Burt Kianian  
Christopher Hylton  
Edward A. Lee  
Xu Liu  
Xiaojun Liu  
Lukho Mikhali  
Steve Neundorfer  
John Reule  
Neil Smith  
Jeff Top  
Brian Vogel  
Whitney Williams  
Yukang Xiong  
Tong Zhao  
Hsiangping Zhang

Department of Electrical Engineering and Computer Sciences  
University of California at Berkeley  
<http://ptolemy.eecs.berkeley.edu>

Document Version 3.0  
In use with Ptolemy II 3.0  
June 8, 2005

Memorandum UCBERE-M0578A

Earlier versions:  
• UCBERE-M0523  
• UCBERE-M0549  
• UCBERE-M0512

This project is supported by the Defense Advanced Research Projects Agency (DARPA), the National Science Foundation, Chou the Center for Hybrid and Embedded Software Systems, the State of California MICRO program, and the following companies: Agilent, Amel, Cadence, Intel, Intelwave, National Semiconductor, Philips, and Wind River Systems.



Volume 2:  
Developer-Oriented



## PTOLEMY II HETEROGENEOUS CONCURRENT MODELING AND DESIGN IN JAVA

Edited by:  
Christopher Hylton, Edward A. Lee, Xu Liu, Xiaojun  
Liu, Steve Neundorfer, Yukang Xiong, Hsiangping Zhang

### VOLUME 3: PTOLEMY II DOMAINS

Authors:  
Shawn S. Bhattacharya  
Elaine Chong  
John Davis, II  
Mehdi Dool  
Burt Kianian  
Christopher Hylton  
Edward A. Lee  
Xu Liu  
Xiaojun Liu  
Lukho Mikhali  
Steve Neundorfer  
John Reule  
Neil Smith  
Jeff Top  
Brian Vogel  
Whitney Williams  
Yukang Xiong  
Tong Zhao  
Hsiangping Zhang

Department of Electrical Engineering and Computer Sciences  
University of California at Berkeley  
<http://ptolemy.eecs.berkeley.edu>

Document Version 3.0  
In use with Ptolemy II 3.0  
June 8, 2005

Memorandum UCBERE-M0578A

Earlier versions:  
• UCBERE-M0523  
• UCBERE-M0549  
• UCBERE-M0512

This project is supported by the Defense Advanced Research Projects Agency (DARPA), the National Science Foundation, Chou the Center for Hybrid and Embedded Software Systems, the State of California MICRO program, and the following companies: Agilent, Amel, Cadence, Intel, Intelwave, National Semiconductor, Philips, and Wind River Systems.



Volume 3:  
Researcher-Oriented

Tutorial information: <http://ptolemy/conferences/07/tutorial.htm>

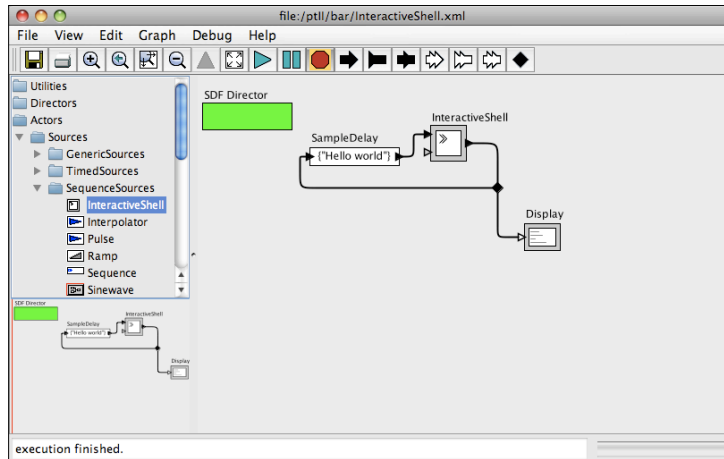
Lee, Berkeley 7

## Outline

- o Building models
- o Models of computation (MoCs)
- o Creating actors
- o Creating directors
- o Software architecture
- o Miscellaneous topics



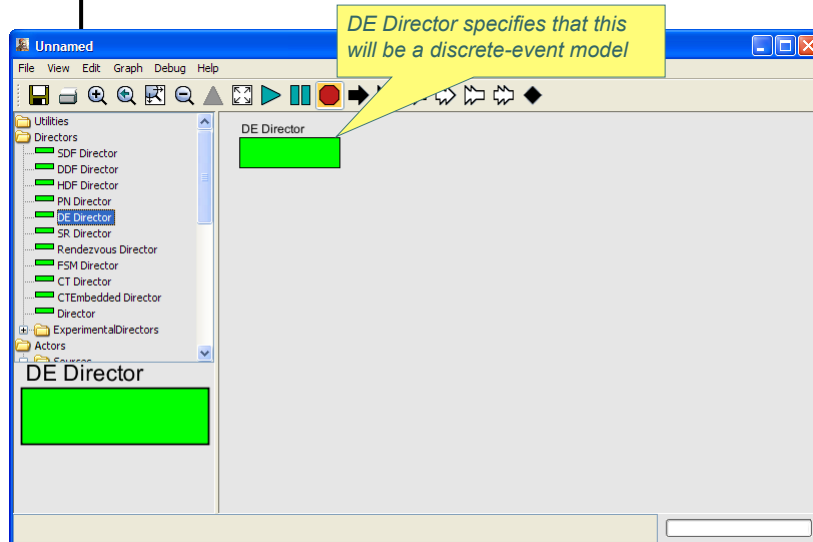
## Building Models – Hello World



Lee, Berkeley 10



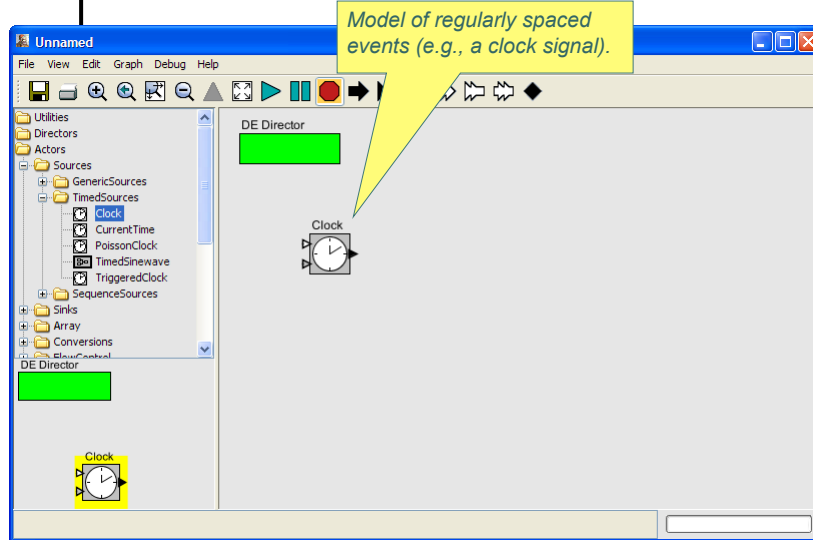
## Building more interesting models



Lee, Berkeley 11



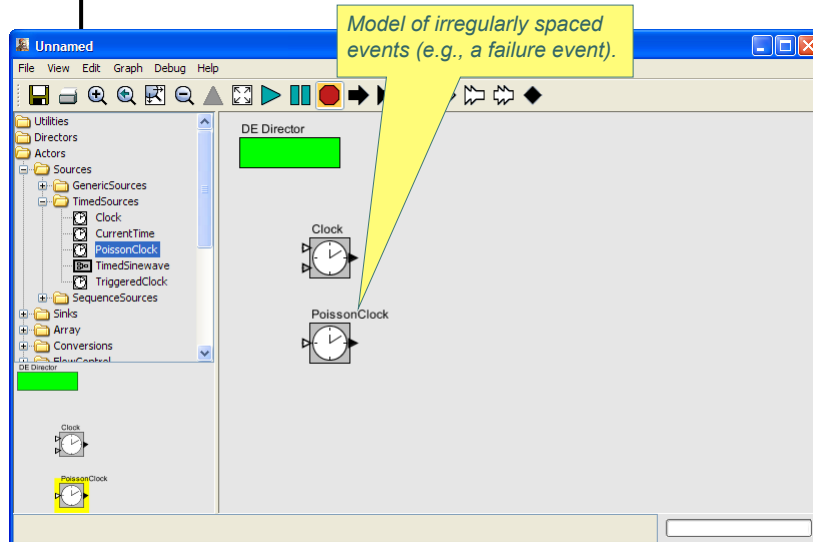
## Building more interesting models



Lee, Berkeley 12

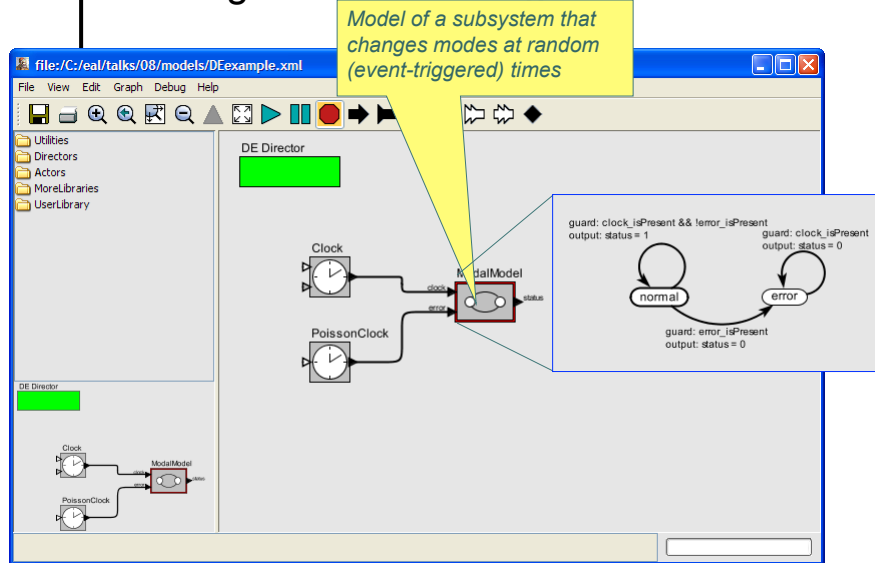


## Building more interesting models



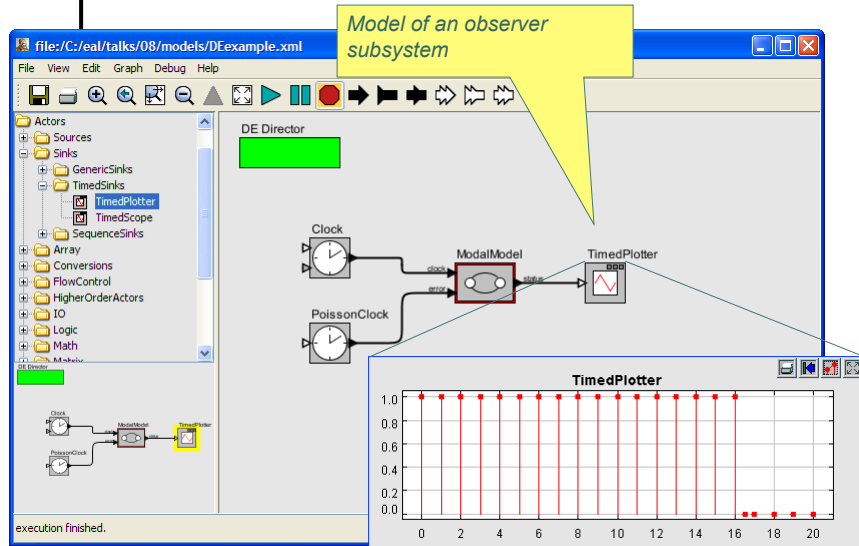
Lee, Berkeley 13

## Building more interesting models



Lee, Berkeley 14

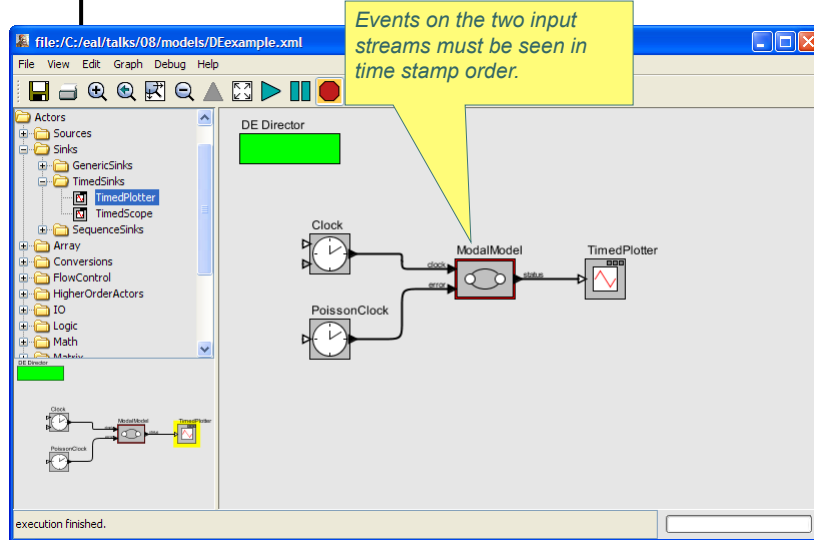
## Building more interesting models



Lee, Berkeley 15



## Building more interesting models

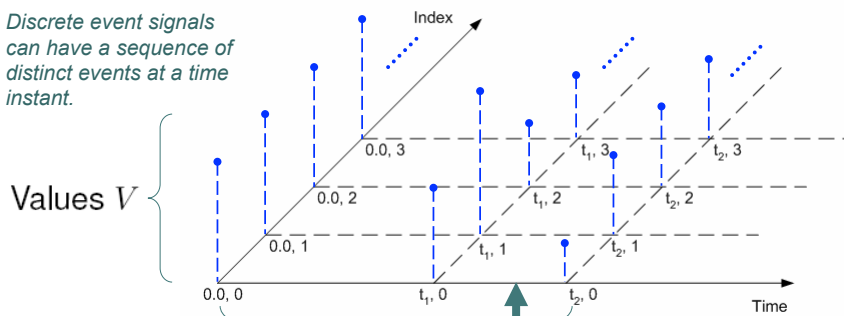


Lee, Berkeley 16



## Ptolemy uses *Superdense* Time

Discrete event signals can have a sequence of distinct events at a time instant.



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

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

Lee, Berkeley 17





## This is a Component Technology

Model of a subsystem given as an imperative program.

```
/** Output the current value.
 * @exception IllegalActionException if there is no director.
 */
public void fire() throws IllegalActionException {
    super.fire();

    // Get the current time and period.
    Time currentTime = getDirector().getModelTime();

    // Indicator whether we've reached the next event.
    _boundaryCrossed = false;

    _tentativeCurrentOutputIndex = _currentOutputIndex;
    output.send(0, _getValue(_tentativeCurrentOutputIndex));

    // In case current time has reached or crossed a boundary to ti
    // next output, update it.
    if (currentTime.compareTo(_nextFiringTime) == 0) {
        _tentativeCurrentOutputIndex++;

        if (_tentativeCurrentOutputIndex >= _length) {
            _tentativeCurrentOutputIndex = 0;
        }

        _boundaryCrossed = true;
    }
}
```



## This is a Component Technology

Model of a subsystem given as a state machine.

```
guard: clock_isPresent && !error_isPresent
output: status = 1

guard: clock_isPresent
output: status = 0

guard: error_isPresent
output: status = 0
```

# This is a Component Technology

Model of a subsystem given as a modal model.

More types of components:

- Modal models
- Functional expressions.
- Submodels in DE
- Submodels in other MoCs

Lee, Berkeley 20

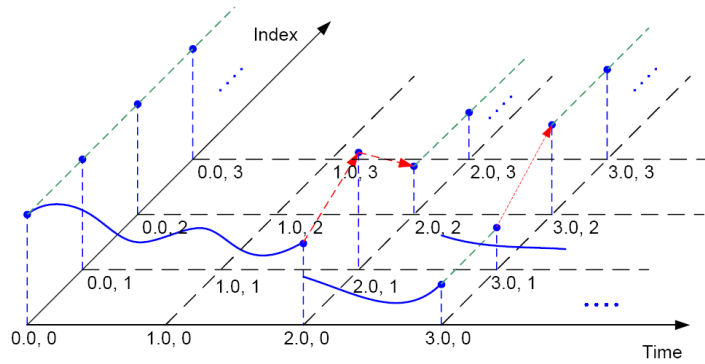
# Continuous-Time Example

Hybrid systems are particularly clean with superdense time. The above signal has multiple values at the times of the transitions.

Lee, Berkeley 21



## Superdense Time for Continuous-Time Signals



At each tag, the signal has **exactly one value**. At each time point, the signal has an **infinite number of values**. The red arrows indicate value changes between tags, which correspond to discontinuities. Signals are **piecewise continuous**, in a well-defined technical sense.

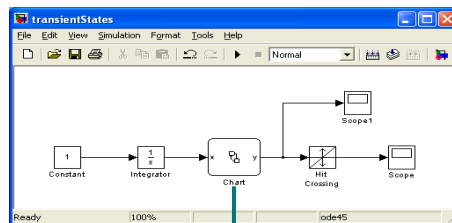
22

Lee, Berkeley 22

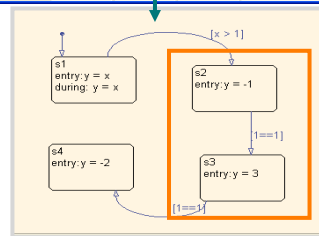


## Contrast with Simulink/Stateflow

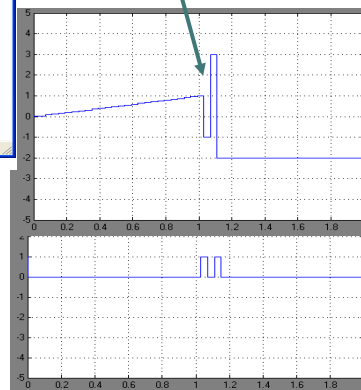
*In Simulink, a signal can only have one value at a given time. Hence Simulink introduces solver-dependent behavior.*



*The simulator engine of Simulink introduces a non-zero delay to consecutive transitions.*



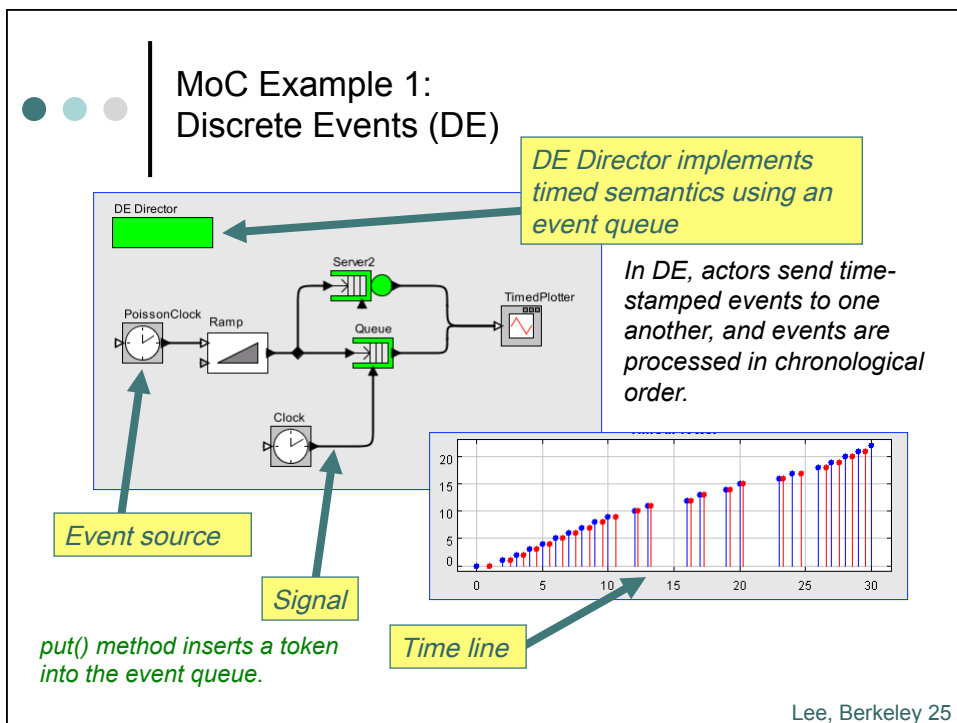
**Transient States**



Lee, Berkeley 23

## Outline

- Building models
- Models of computation (MoCs)
- Creating actors
- Creating directors
- Software architecture
- Miscellaneous topics



## MoC Example 2: Kahn Process Networks (PN)

**PN Director**

This model, whose structure is due to Kahn and MacQueen, calculates integers whose prime factors are only 2, 3, and 5, with no redundancies. It uses the OrderedMerge actor, which takes two monotonically increasing input sequences and merges them into one monotonically increasing output sequence.

In the PN domain, each actor executes in its own Java thread. That thread iteratively reads inputs, performs computation, and produces outputs.

*Kahn, MacQueen, 1977*

The output is an ordered sequence of integers of the form  $2^n * 3^m * 5^k$ , where n, m, and k are non-negative integers.

*actor == thread*

*signal == stream*

*reads block*

*writes don't*

This BooleanSwitch is used to starve the model after a power of 5 greater than 1000000 is produced. This results in deterministically stopping the execution.

Lee, Berkeley 26

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

## MoC 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**

**Synchronous Dataflow Modeling**

Estimate the spectrum of three sinusoids in noise by three different techniques.

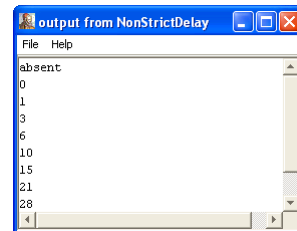
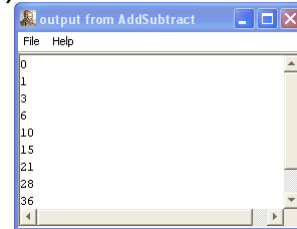
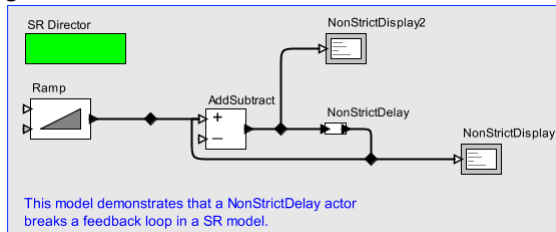
This example illustrates SDF modeling, which is well-suited to signal processing. In SDF, components communicate using streams, but their production and consumption rates are fixed. Because of these fixed rates, extensive static analysis of the model is possible, enabling efficient code generation and optimization.

*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 27

## MoC Example 4: Synchronous/Reactive (SR)

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

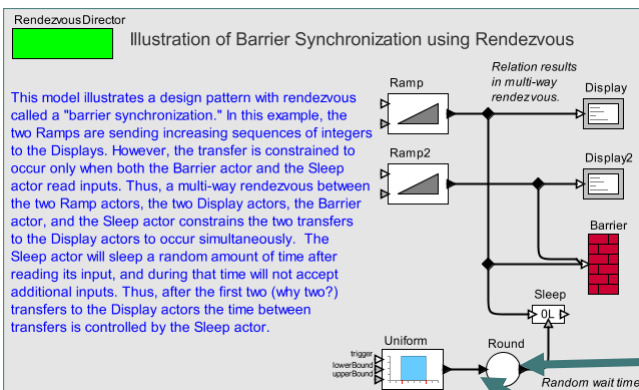


Like SDF, SR is decidable and suitable for code generation. It is harder to parallelize than SDF, however.

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

Lee, Berkeley 28

## MoC 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.

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

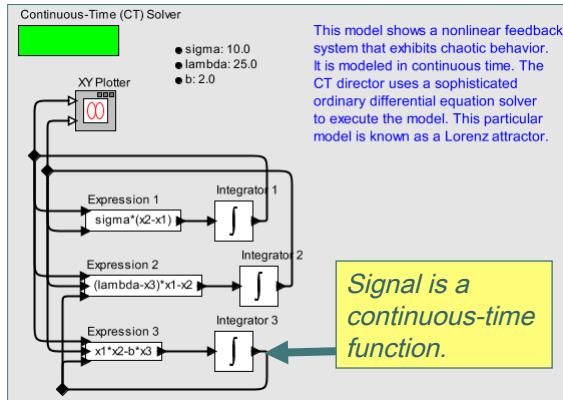
actor == thread

reads block

writes block

Lee, Berkeley 29

## MoC Example 6: Continuous Time (CT)

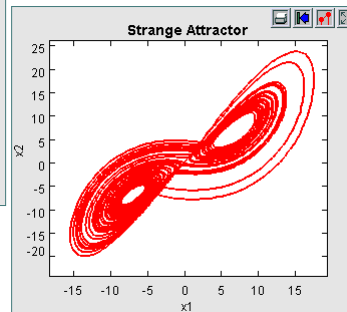


This model shows a nonlinear feedback system that exhibits chaotic behavior. It is modeled in continuous time. The CT director uses a sophisticated ordinary differential equation solver to execute the model. This particular model is known as a Lorenz attractor.

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

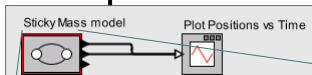
Signal is a continuous-time function.

Director includes an ODE solver.



Lee, Berkeley 30

## Ptolemy II Hierarchy Supports Heterogeneity



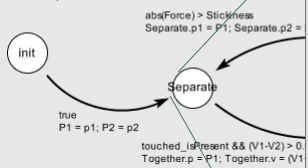
Concurrent actors governed by one model of computation (e.g., Discrete Events).

Modal behavior given in another MoC.

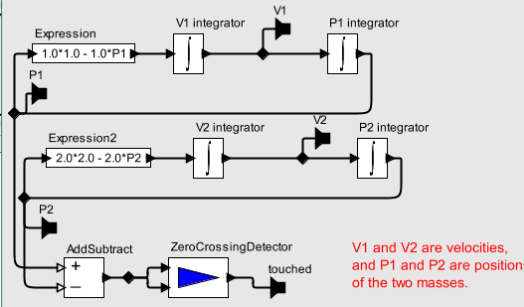
The sticky masses system has two modes of operation, "Separate" and "Together," corresponding to the point masses are stuck together. The "init" has a transition that is used to initialize the "Separate" model (double click on that transition to see its

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 31

## Outline

- Building models
- Models of computation (MoCs)
- Creating actors
- Creating directors
- Software architecture
- Miscellaneous topics

● ● ● Actors



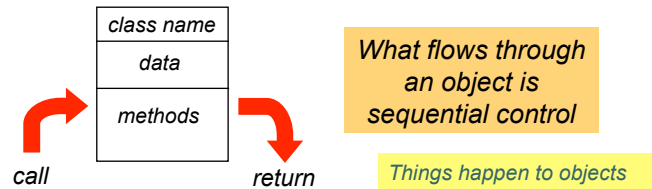
Lee, Berkeley 35



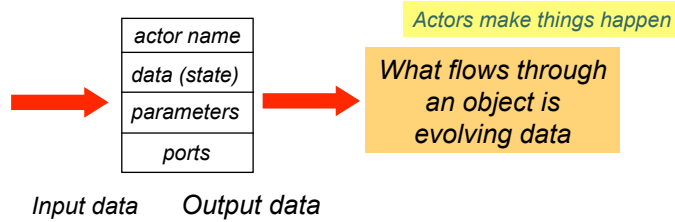


## Ptolemy Components are Actors and Objects

*The established: Object-oriented:*



*The alternative: Actor oriented:*

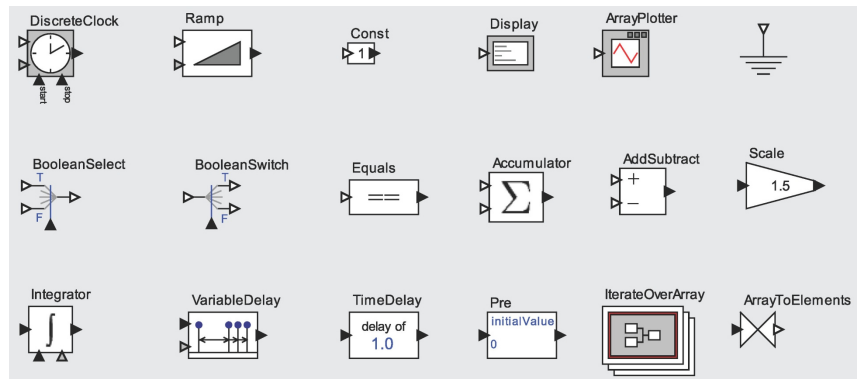


Lee, Berkeley 36



## Actors

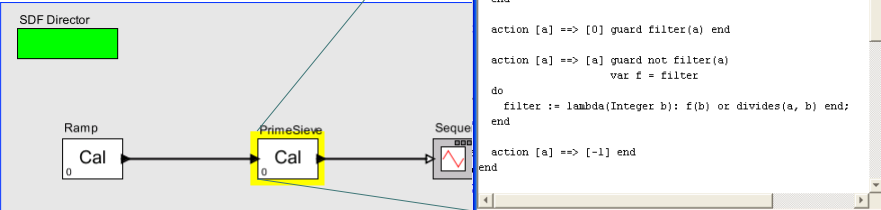
- Ptolemy has a library of predefined actors
- Java classes that implement the “executable” interface



Lee, Berkeley 37

## Actors can be defined in Java, C, Python, Cal, and MATLAB

Cal, developed by Joern Janneck (now at Lund) is a language for defining actors that are analyzable for key behavioral properties.



This model demonstrates the use of function closures inside a CAL actor.

The PrimeSieve actor uses nested function closures to realize the Sieve of Eratosthenes, a method for finding prime numbers. Its state variable, "filter," contains the current filter function. If it is "false" a new prime number has been found, and a new filter function will be generated.

The PrimeSieve actor expects an ascending sequence of natural numbers, starting from 2, as input.

Lee, Berkeley 38

## Most Actors are Written in Java

The screenshot shows the Ptolemy II environment. On the left, a tree view shows the 'Actors' directory. In the center, a model diagram shows a 'Gaussian' actor connected to a 'String Sequence' actor. On the right, a code window shows the Java implementation of the Gaussian actor:

```

public class Gaussian extends RandomSource {
  /** Construct an actor with the given container and name.
   * @param container The container.
   * @param name The name of this actor.
   * @exception IllegalArgumentException If the actor cannot be contained
   *   by the proposed container.
   * @exception NameDuplicationException If the container already has an
   *   actor with this name.
   */
  public Gaussian(CompositeEntity container, String name)
    throws NameDuplicationException, IllegalArgumentException {
    super(container, name);
    output.setTypeEquals(BaseType.DOUBLE);
    mean = new PortParameter(this, "mean", new DoubleToken(0.0));
    mean.setTypeEquals(BaseType.DOUBLE);
    standardDeviation = new PortParameter(this, "standardDeviation");
    standardDeviation.setExpression("1.0");
    standardDeviation.setTypeEquals(BaseType.DOUBLE);
  }
  ///////////////////////////////////////////////////
  // ports and parameters
  ///////////////////////////////////////////////////
  /** The mean of the random number.
   * as type double, initially with value 0.
   */
  PortParameter mean;
  /** standard deviation of the random number.
   * as type double, initially with value 1.
   */
  PortParameter standardDeviation;
  ///////////////////////////////////////////////////
  // public methods
  ///////////////////////////////////////////////////
}
  
```

A context menu is open over the Gaussian actor, showing options like 'Customize', 'Documentation', 'Appearance', 'Save Actor In Library', 'Listen to Actor', 'Set Breakpoints', 'Convert to Class', 'Open Actor', and 'Open Instance'.

Berkeley 39



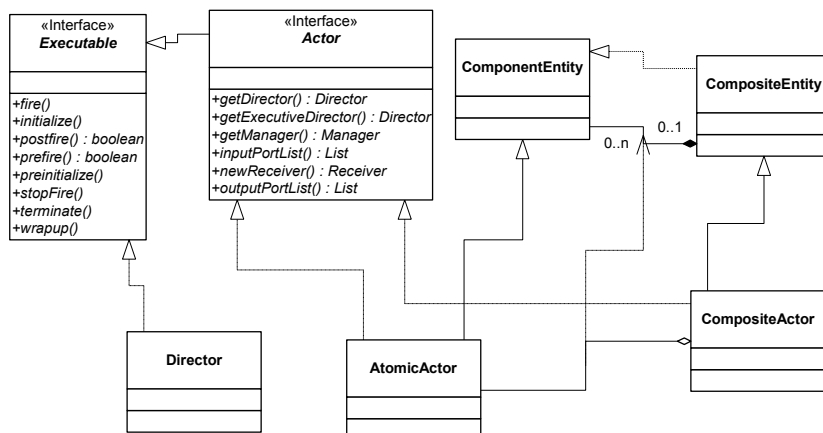
## Simple String Manipulation Actor in Java

```
public class Ptoleimizer extends TypedAtomicActor {
    public Ptoleimizer(CompositeEntity container, String name)
        throws IllegalArgumentException, NameDuplicationException {
        super(container, name);
        input = new TypedIOPort(this, "input");
        input.setTypeEquals(BaseType.STRING);
        input.setInput(true);
        output = new TypedIOPort(this, "output");
        output.setTypeEquals(BaseType.STRING);
        output.setOutput(true);
    }
    public TypedIOPort input;
    public TypedIOPort output;
    public void fire() throws IllegalArgumentException {
        if (input.hasToken(0)) {
            Token token = input.get(0);
            String result = ((StringToken)token).stringValue();
            result = result.replaceAll("t", "pt");
            output.send(0, new StringToken(result));
        }
    }
}
```

Lee, Berkeley 40



## Object Model for Executable Components



Lee, Berkeley 41



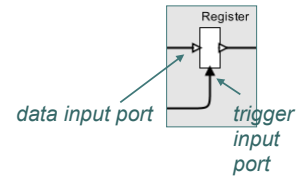
## Definition of the Register Actor (Sketch)

```
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.*

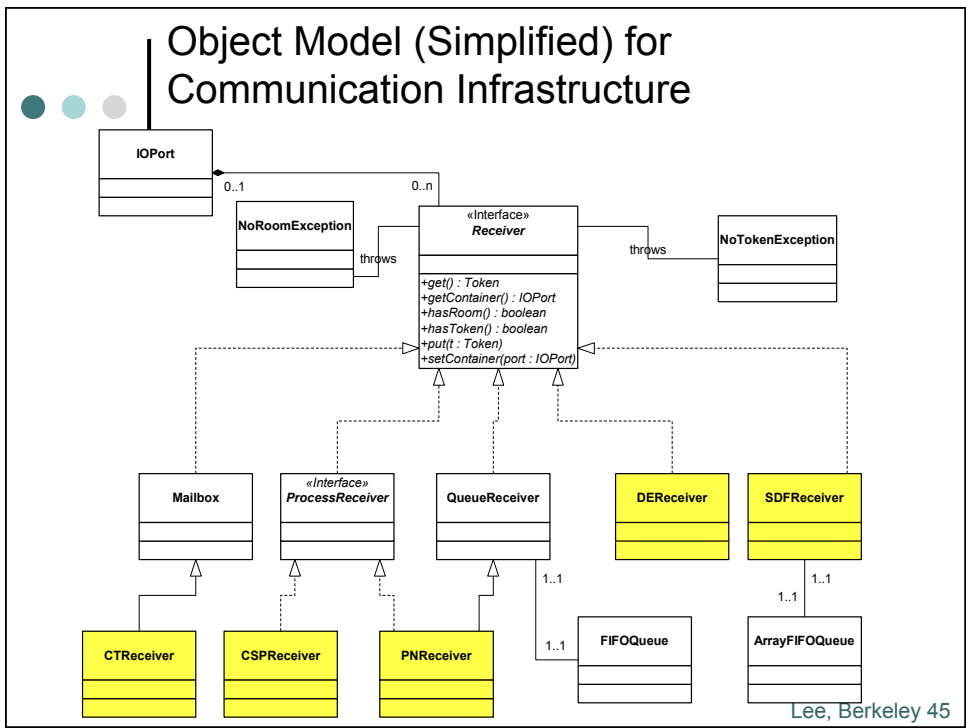
*Read the data input and update the state.*



Lee, Berkeley 42

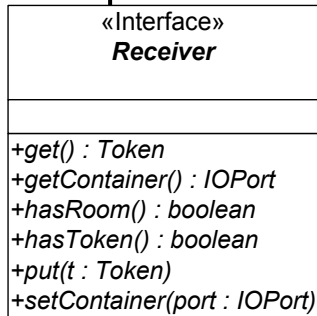
## Outline

- Building models
- Models of computation (MoCs)
- Creating actors
- Creating directors
- Software architecture
- Miscellaneous topics



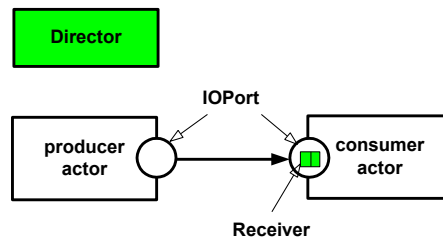


## Object-Oriented Approach to Achieving Behavioral Polymorphism



*These polymorphic methods implement the communication semantics of a domain in Ptolemy II. The receiver instance used in communication is supplied by the director, not by the component.*

**Recall: Behavioral polymorphism** is the idea that components can be defined to operate with multiple models of computation and multiple middleware frameworks.



Lee, Berkeley 46



## Extension Exercise

Build a director that subclasses PNDirector to allow ports to alter the “blocking read” behavior. In particular, if a port has a parameter named “tellTheTruth” then the receivers that your director creates should “tell the truth” when hasToken() is called. That is, instead of always returning true, they should return true only if there is a token in the receiver.

Parameterizing the behavior of a receiver is a simple form of communication refinement, a key principle in, for example, Metropolis.

Lee, Berkeley 47



## Implementation of the NondogmaticPNDirector

```
package doc.tutorial;
import ...
public class NondogmaticPNDirector extends PNDirector {
    public NondogmaticPNDirector(CompositeEntity container, String name)
        throws IllegalArgumentException, NameDuplicationException {
        super(container, name);
    }
    public Receiver newReceiver() {
        return new FlexibleReceiver();
    }
    public class FlexibleReceiver extends PNQueueReceiver {
        public boolean hasToken() {
            IOPort port = getContainer();
            Attribute attribute = port.getAttribute("tellTheTruth");
            if (attribute == null) {
                return super.hasToken();
            }
            // Tell the truth...
            return _queue.size() > 0;
        }
    }
}
```

Lee, Berkeley 48



## Using It

**With NondogmaticPNDirector:**

**With PNDirector:**

**SDF Director**  
Model of a sensor sensing a sinusoidal signal with the specified frequency and phase at the specified sampling frequency. This composite actor simulates real-time behavior by sleeping the amount of time given by the samplingPeriod (in seconds) before producing an output.

Legend:  
• fre  
• ph  
• sa  
• no

Lee, Berkeley 49



## Designing a Sensible MoC is not so easy! Consider Kahn Process Networks (PN)

- A set of components called *actors*.
- Each representing a sequential procedure.
- Where steps in these procedures receive or send messages to other actors (or perform local operations).
- Messages are communicated asynchronously with unbounded buffers.
- A procedure can always send a message. It does not need to wait for the recipient to be ready to receive.
- Messages are delivered reliably and in order.
- When a procedure attempts to receive a message, that attempt blocks the procedure until a message is available.

Lee, Berkeley 50



## Coarse History

- Semantics given by Gilles Kahn in 1974.
  - Fixed points of continuous and monotonic functions
- More limited form given by Kahn and MacQueen in 1977.
  - Blocking reads and nonblocking writes.
- Generalizations to nondeterministic systems
  - Kosinski [1978], Stark [1980s], ...
- Bounded memory execution given by Parks in 1995.
  - Solves an undecidable problem.
- Debate over validity of this policy, Geilen and Basten 2003.
  - Relationship between denotational and operational semantics.
- Many related models intertwined.
  - Actors (Hewitt, Agha), CSP (Hoare), CCS (Milner), Interaction (Wegner), Streams (Broy, ...), Dataflow (Dennis, Arvind, ...)...

Lee, Berkeley 51





## Dataflow

Dataflow models are similar to PN models except that actor behavior is given in terms of discrete “firings” rather than processes. A firing occurs in response to inputs.

Lee, Berkeley 54



## A few variants of dataflow MoCs

- *Computation graphs* [Karp and Miller, 1966]
- *Static dataflow* [Dennis, 1974]
- *Dynamic dataflow* [Arvind, 1981]
- *Structured dataflow* [Matwin & Pietrzykowski 1985]
- *K-bounded loops* [Culler, 1986]
- *Synchronous dataflow* [Lee & Messerschmitt, 1986]
- *Structured dataflow and LabVIEW* [Kodosky, 1986]
- *PGM: Processing Graph Method* [Kaplan, 1987]
- *Synchronous languages* [Lustre, Signal, 1980's]
- *Well-behaved dataflow* [Gao, 1992]
- *Boolean dataflow* [Buck and Lee, 1993]
- *Multidimensional SDF* [Lee, 1993]
- *Cyclo-static dataflow* [Lauwereins, 1994]
- *Integer dataflow* [Buck, 1994]
- *Bounded dynamic dataflow* [Lee & Parks, 1995]
- *Heterochronous dataflow* [Girault, Lee, & Lee, 1997]
- *Scenarios* [Geilen & Stuijk, 2010]
- ...

Lee, Berkeley 55



## Some Subtleties

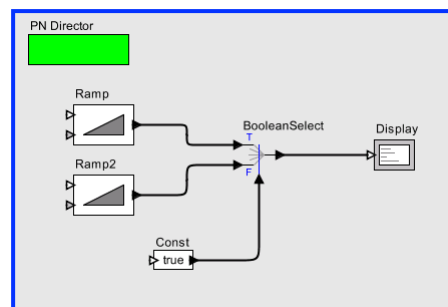
- Termination, deadlock, and livelock (halting)
- Bounding the buffers.
- Fairness
- Parallelism
- Data structures and shared data
- Determinism
- Real-time constraints
- Syntax

Lee, Berkeley 56



## Question 1: Is “Fair” Scheduling a Good Idea?

In the following model, what happens if every actor is given an equal opportunity to run?

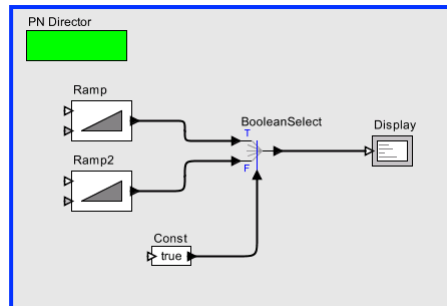


Lee, Berkeley 57



## Question 2: Is “Data-Driven” Execution a Good Idea?

In the following model, if actors are allowed to run when they have input data on connected inputs, what will happen?

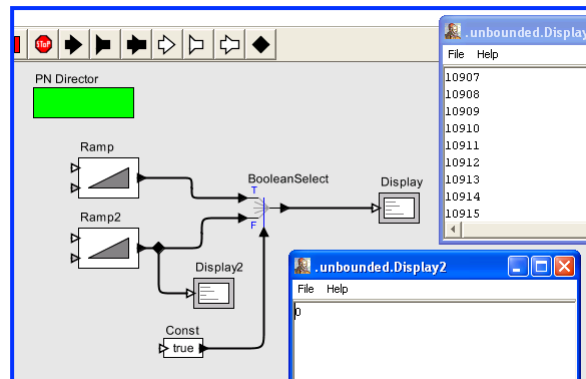


Lee, Berkeley 58



## Question 3: When are Outputs Required?

Is the execution shown for the following model the “right” execution?

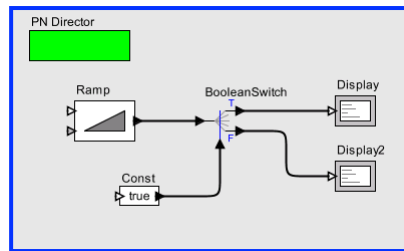


Lee, Berkeley 59



## Question 4: Is “Demand-Driven” Execution a Good Idea?

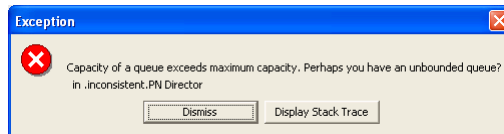
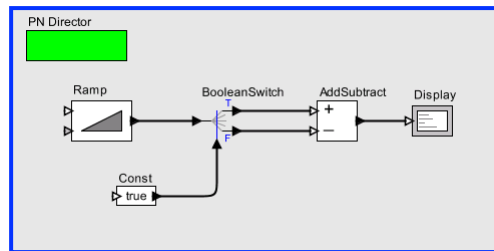
In the following model, if actors are allowed to run when another actor requires their outputs, what will happen?



Lee, Berkeley 60



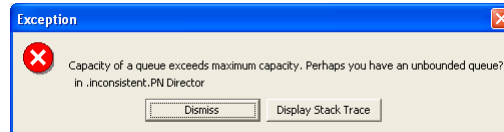
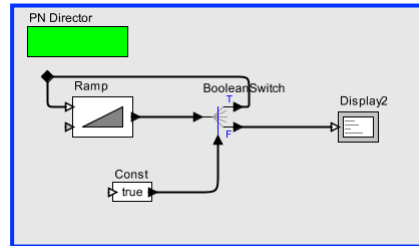
## Question 5: What is the “Correct” Execution of This Program?



Lee, Berkeley 61



## Question 6: What is the Correct Behavior of this Program?



Lee, Berkeley 62



## Naïve Schedulers Fail

- Fair
- Demand driven
- Data driven
- Most mixtures of demand and data driven

*If people insist on building their own MoCs from scratch, what will keep them from repeating the mistakes that have been made by top experts in the field?*

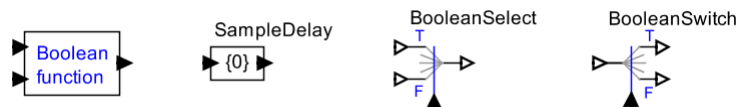
Lee, Berkeley 63



## Programmers should not have to figure out how to solve these problems!

*Undecidability and Turing Completeness* [Buck 93]

Given the following four actors and Boolean streams, you can construct a universal Turing machine:



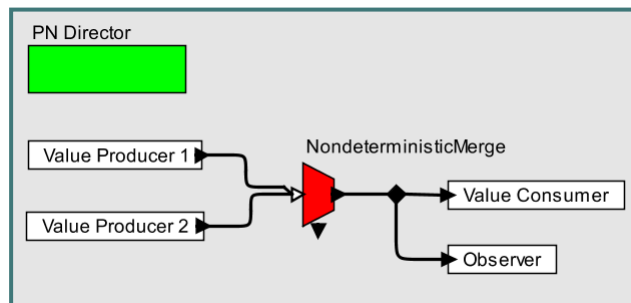
Hence, the following questions are undecidable:

- Will a model deadlock (terminate)?
- Can a model be executed with bounded buffers?

Lee, Berkeley 64



## Question 7: How to support nondeterminism?



Merging of streams is needed for some applications. Does this require fairness? What does fairness mean?

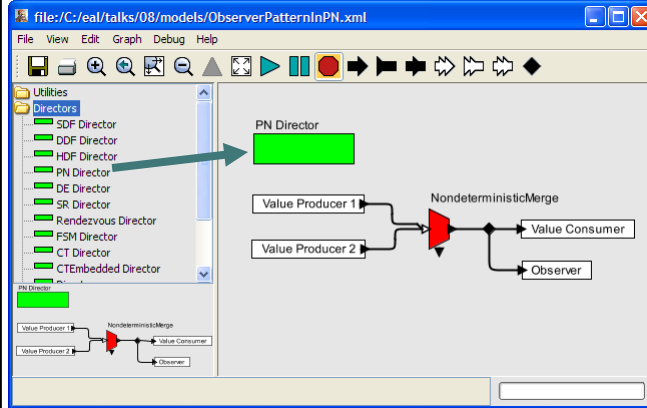
Lee, Berkeley 65



These problems have been solved!  
*Let's not make programmers re-solve them for every program.*

Library of directors

Program using actor-oriented components and a PN MoC



Directors should be designed by experts in languages and concurrency, not by application model builders.

Lee, Berkeley 66



The PN Director solves the above problems by implementing a “useful execution”

Define a **correct execution** to be any execution for which after any finite time every signal is a prefix of the signal given by the (Kahn) least-fixed-point semantics.

Define a **useful execution** to be a correct execution that satisfies the following criteria:

1. For every non-terminating model, after any finite time, a useful execution will extend at least one stream in finite (additional) time.
2. If a correct execution satisfying criterion (1) exists that executes with bounded buffers, then a useful execution will execute with bounded buffers.

Lee, Berkeley 67



## Our solution: Parks' Strategy [Parks 95]

### This “solves” the undecidable problems:

- Start with an arbitrary bound on the capacity of all buffers.
- Execute as much as possible.
- If deadlock occurs and at least one actor is blocked on a write, increase the capacity of at least one buffer to unblock at least one write.
- Continue executing, repeatedly checking for deadlock.

This delivers a useful execution (possibly taking infinite time to tell you whether a model deadlocks and how much buffer memory it requires).

Lee, Berkeley 68



## There are many more subtleties!

*We need disciplined concurrent models of computation, not arbitrarily flexible libraries.*

Some principles:

- Do not use nondeterministic programming models to accomplish deterministic ends.
- Use concurrency models that have analogies in the physical world (actors, not threads).
- Provide these in the form of models of computation (MoCs) with well-developed semantics and tools.
- Use specialized MoCs to exploit semantic properties (avoid excess generality).
- Leave the choice of shared memory or message passing to the compiler.

Lee, Berkeley 69





## Extension Exercise 2

Build a director that subclasses Director and allows different receiver classes to be used on different connections. This is a form of what we call “amorphous heterogeneity.”

We will not do this today.  
See `$PTII/doc/tutorial/domains`

Lee, Berkeley 70



## Extension Exercise 3

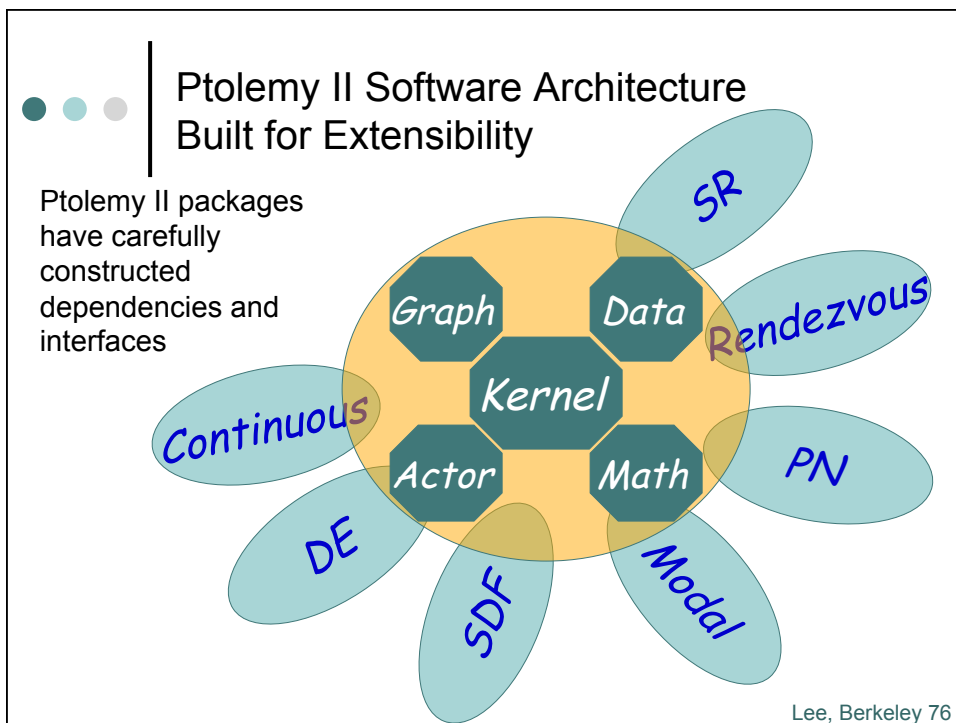
Build a director that fires actors in left-to-right order, as they are laid out on the screen.

We will not do this today.  
See `$PTII/doc/tutorial/domains`

Lee, Berkeley 73

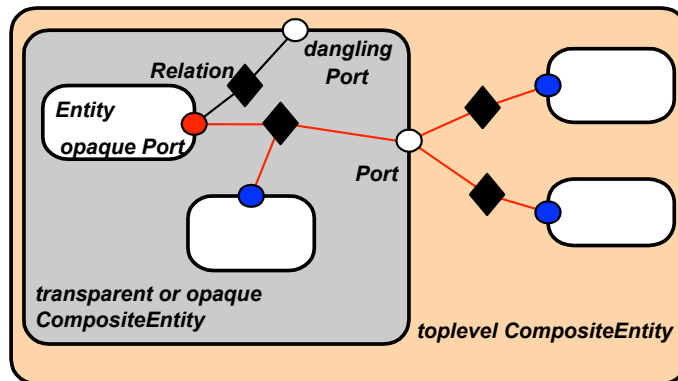
## Outline

- Building models
- Models of computation (MoCs)
- Creating actors
- Creating directors
- Software architecture
- Miscellaneous topics





## Hierarchy - Composite Components



Lee, Berkeley 77

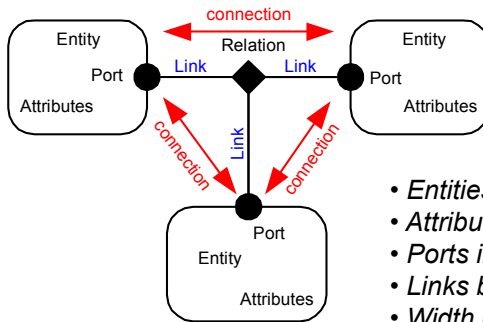


## Separable Tool Architecture

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

Lee, Berkeley 78

## The Basic Abstract Syntax for Composition



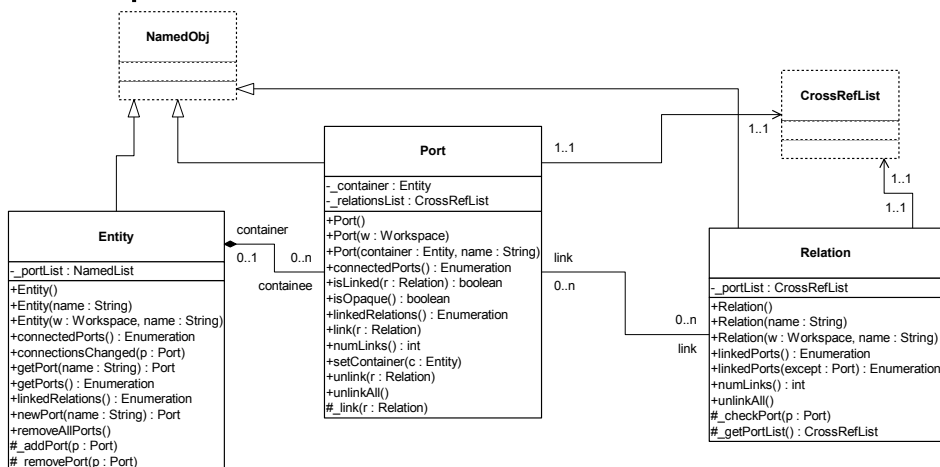
- Entities
- Attributes on entities (parameters)
- Ports in entities
- Links between ports
- Width on links (channels)
- Hierarchy

Concrete syntaxes:

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

Lee, Berkeley 79

## Meta Model: Kernel Classes Supporting the Abstract Syntax



*These get subclassed for specific purposes.*

Lee, Berkeley 80



## Separable Tool Architecture

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

Lee, Berkeley 81



## 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 82



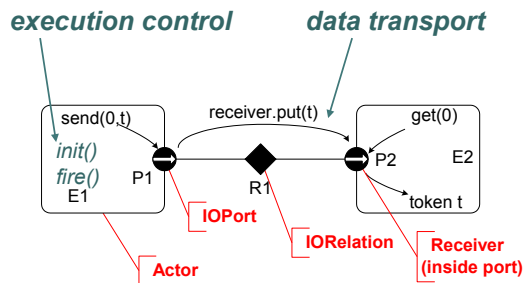
## Separable Tool Architecture

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

Lee, Berkeley 83



## 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)
- ...

Lee, Berkeley 84



## Implemented as a Java interface

### Interface "Executable"

Method Summary	
void	<b>fire()</b> Fire the actor.
boolean	<b>isFireFunctional()</b> Return true if this executable does not change state in either the prefire() or the fire() method.
boolean	<b>isStrict()</b> Return true if this executable is strict, meaning all inputs must be known before iteration.
int	<b>iterate(int count)</b> Invoke a specified number of iterations of the actor.
boolean	<b>postfire()</b> This method should be invoked once per iteration, after the last invocation of fire() in that iteration.
boolean	<b>prefire()</b> This method should be invoked prior to each invocation of fire().
void	<b>stop()</b> Request that execution of this Executable stop as soon as possible.
void	<b>stopFire()</b> Request that execution of the current iteration complete.
void	<b>terminate()</b> Terminate any currently executing model with extreme prejudice.

Lee, Berkeley 85



## Example execution sequence

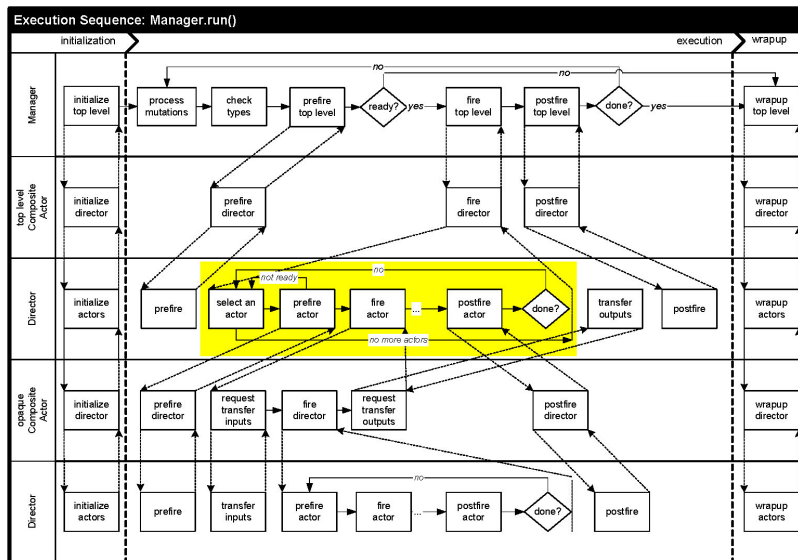
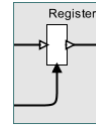


FIGURE 2.14. Example execution sequence implemented by run() method of the Director class.

Lee, Berkeley 86



## How Does This Work? Execution of Ptolemy II Actors



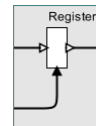
Flow of control:

- Initialization
- Execution
- Finalization

Lee, Berkeley 87



## How Does This Work? Execution of Ptolemy II Actors



Flow of control:

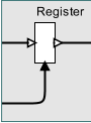
- Initialization
- Execution
- Finalization

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

Lee, Berkeley 88




How Does This Work?  
Execution of Ptolemy II Actors



Flow of control:

- Initialization
- **Execution**
- Finalization

*Iterate*



```

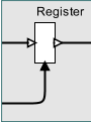
If (prefire()) {
    fire();
    postfire();
}

```

*Only the postfire() method should change the state of the actor.*

Lee, Berkeley 89

How Does This Work?  
Execution of Ptolemy II Actors



Flow of control:

- Initialization
- Execution
- **Finalization**

Lee, Berkeley 90



## Definition of the Register Actor (Sketch)

Can the actor fire?

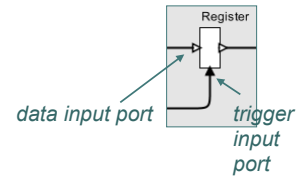
```
class Register extends TypedAtomicActor {
  private Object state;
  boolean prefire() {
    if (trigger is known) { return true; }
  }
}
```

React to trigger input.

```
void fire() {
  if (trigger is present) {
    send state to output;
  } else {
    assert output is absent;
  }
}
```

Read the data input and update the state.

```
void postfire() {
  if (trigger is present) {
    state = value read from data input;
  }
}
```



Lee, Berkeley 91



## Separable Tool Architecture

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

Lee, Berkeley 92



## 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 93

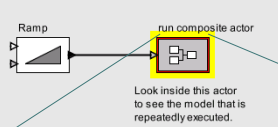
## Outline

- Building models
- Models of computation (MoCs)
- Creating actors
- Creating directors
- Software architecture
- Miscellaneous topics

## Example Extensions Using Models to Control Models

**SDF**

This model illustrates the use of a "run composite actor" component. That component contains another Ptolemy II model. Each time it fires, it performs a complete execution of that other Ptolemy II model, rather than just one firing as would be typical of a composite actor.s



Ramp → run composite actor

Look inside this actor to see the model that is repeatedly executed.

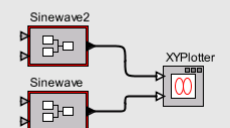
---

**SDF**

This model generates Lissajous figures, which are plots of one sinusoid vs. another. On each execution, it generates one figure.

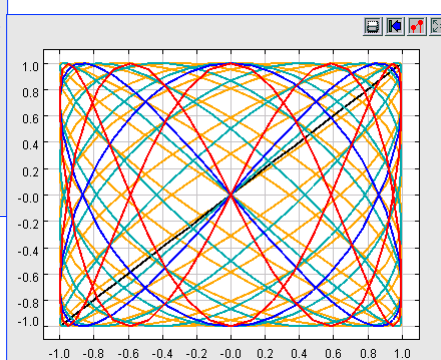
run: 1

This "port parameter" provides a way to get inputs to the model where the value differs on each run.



Sinewave2 → XYPlotter

Sinewave → XYPlotter

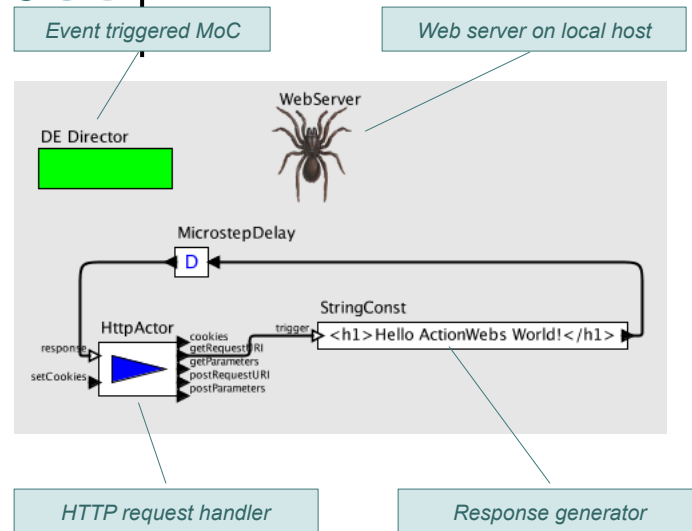


This is an example of a "higher-order component," or an actor that references one or more other actors.

MultipleRuns SDF demo

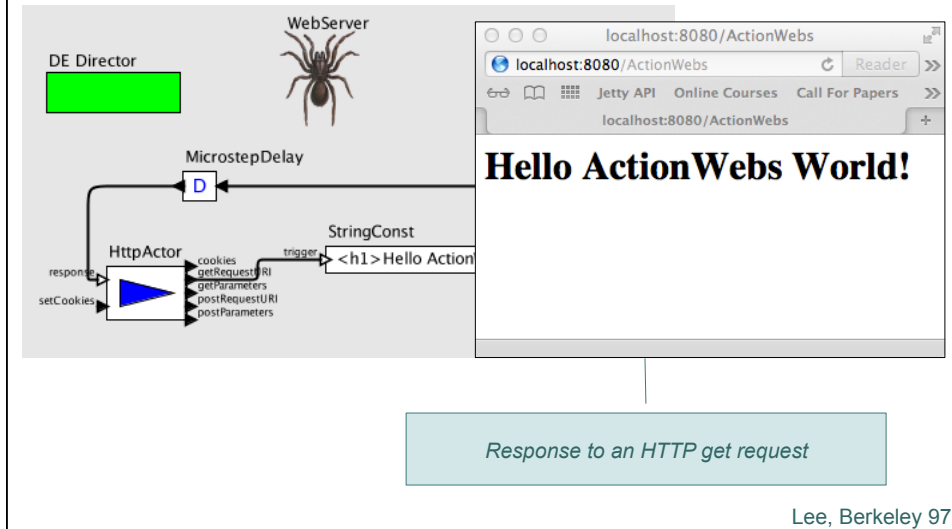
Lee, Berkeley 95

## Example Extensions: Rapid Construction of Action Web Services



Lee, Berkeley 96

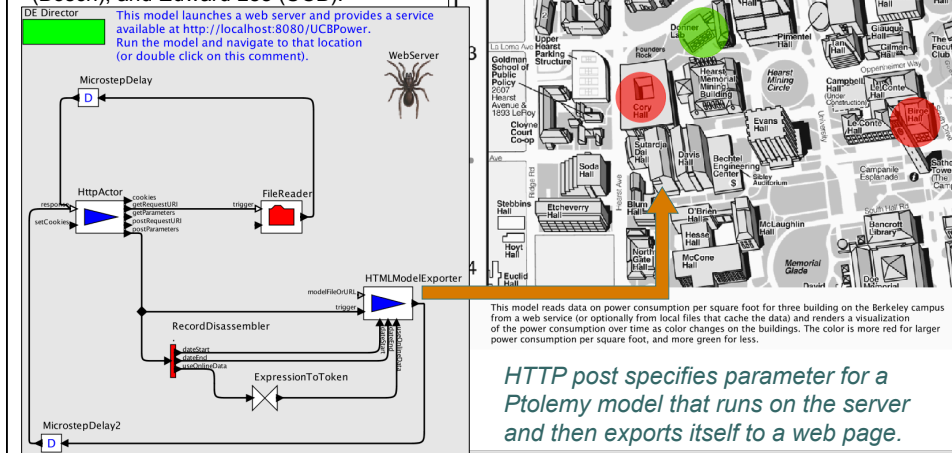
## Example Extensions: Rapid Construction of Action Web Services



## Example: UCB Power

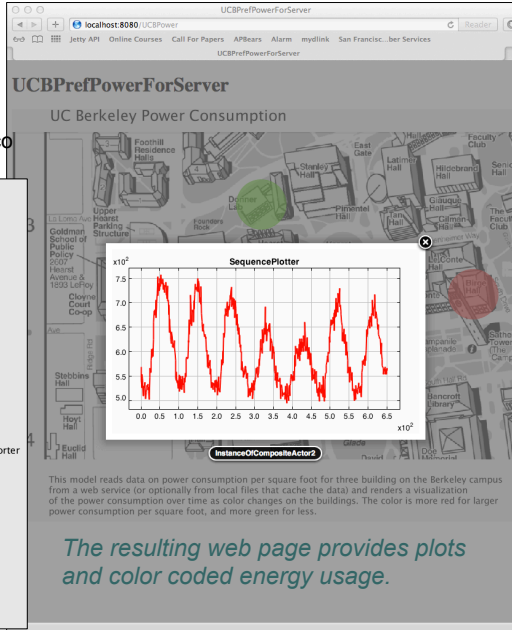
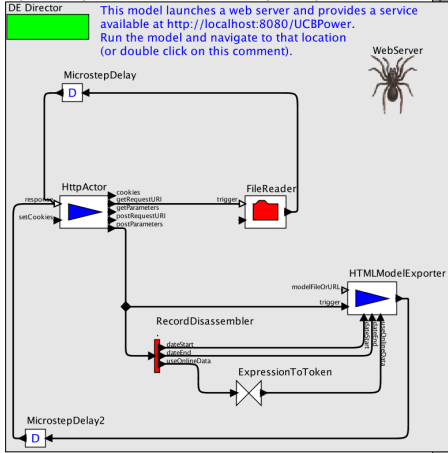
This model retrieves energy data from OpenBMS (LoCal) and displays it. Authors: Roxana Gheorghiu (Pitt), Elizabeth Latronico (Bosch), and Edward Lee (UCB).

This model launches a web server and provides a service available at <http://localhost:8080/UCBPower>. Run the model and navigate to that location (or double click on this comment).



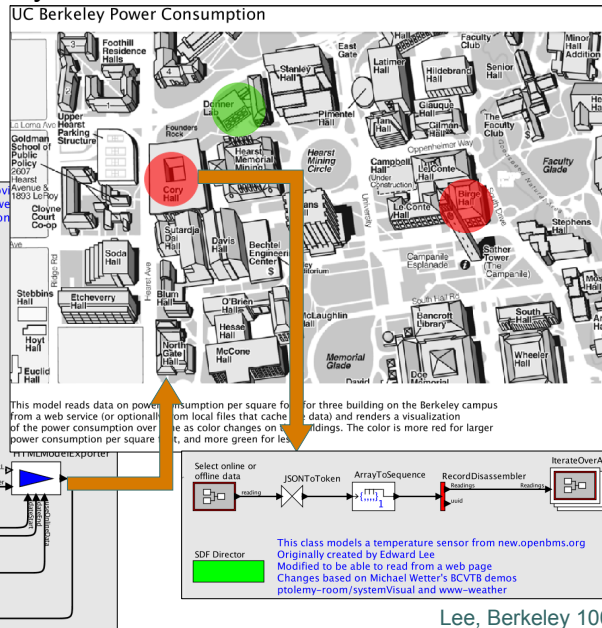
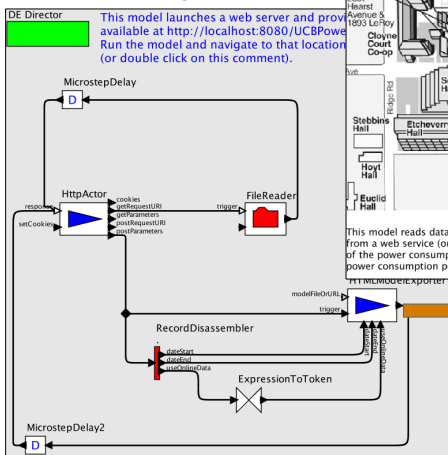
## Example: UCB Power

This model retrieves energy data from OpenBMS (LoCal) and displays it. Authors: Roxana Gheorghiu (Pitt), Elizabeth Latronico (Bosch), and Edward Lee (UCB).



## Under the Hood: Ptolemy model runs and exports another Ptolemy model.

HTMLModelExporter references another model, shown on the right.





## Ptolemy II Extension Points

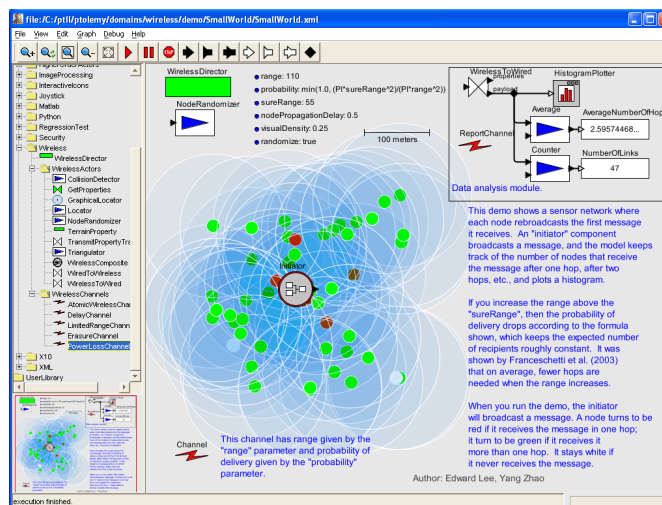
- Define actors
- Interface to foreign tools (e.g. Python, MATLAB)
- Interface to verification tools (e.g. Chic)
- Define actor definition languages
- Define directors (and models of computation)
- Define visual editors
- Define textual syntaxes and editors
- Packaged, branded configurations

All of our “domains” are extensions built on a core infrastructure.

Lee, Berkeley 102



## Extension of Discrete-Event Modeling for Wireless Sensor Nets



*VisualSense extends the Ptolemy II discrete-event domain with communication between actors representing sensor nodes being mediated by a channel, which is another actor.*

*The example at the left shows a grid of nodes that relay messages from an initiator (center) via a channel that models a low (but non-zero) probability of long range links being viable.*

Lee, Berkeley 103

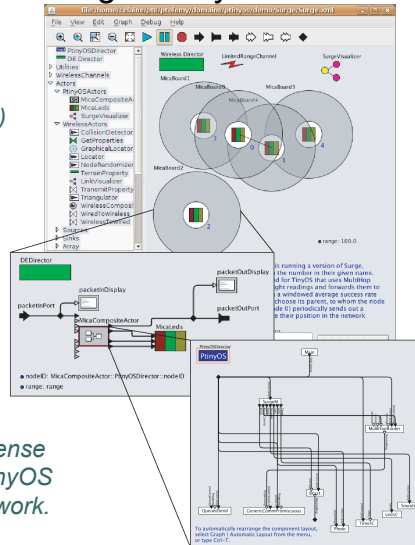
## Viptos: Extension of VisualSense with Programming of TinyOS nodes

Viptos demo:  
Multihop routing (Surge)



Hardware

Viptos extends VisualSense with programming of TinyOS nodes in a wireless network. See the Ph.D. thesis of Elaine Cheong (Aug 2007).



Physical environment  
Simulation  
(with visualization of routing tree)

Software  
Code generation:  
Models to nesC.

Lee, Berkeley 104

## Another Extension: HyVisual – Hybrid System Modeling Tool Based on Ptolemy II

**Refinement Solver**

This models the dynamics of a ball falling in a gravitational field.

HyVisual 2.2-beta - Hybrid System Visual Modeler

Block-diagram editor and simulator for hybrid systems.

- Documentation
- Copyright

To start immediately by creating a hybrid system, select File, New, Graph Editor from the menu bar. Select Help from the Help menu for instructions on creating a model.

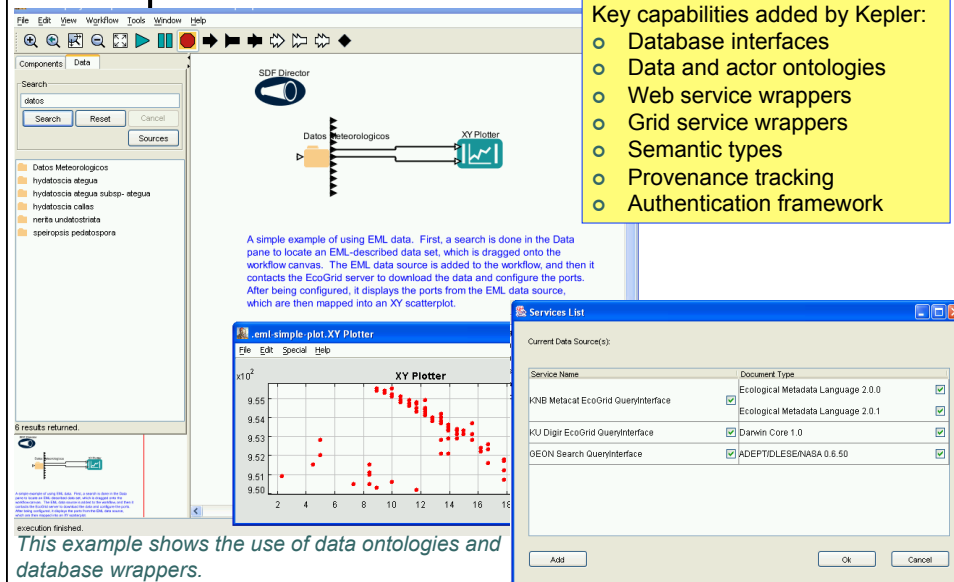
Hybrid systems are systems with continuous time dynamics, discrete events, and discrete mode changes. This visual modeler supports construction of hierarchical hybrid systems. It uses a block-diagram representation of ordinary differential equations (ODE) to define continuous dynamics. It uses a bubble-and-gate diagram representation of finite

HyVisual was first released in January 2003.

Lee, Berkeley 105



## Another Extension: Kepler: Aimed at Scientific Workflows



**Key capabilities added by Kepler:**

- Database interfaces
- Data and actor ontologies
- Web service wrappers
- Grid service wrappers
- Semantic types
- Provenance tracking
- Authentication framework

A simple example of using EML data. First, a search is done in the Data pane to locate an EML-described data set, which is dragged onto the workflow canvas. The EML data source is added to the workflow, and then it contacts the EcoGrid server to download the data and configure the ports. After being configured, it displays the ports from the EML data source, which are then mapped into an XY scatterplot.

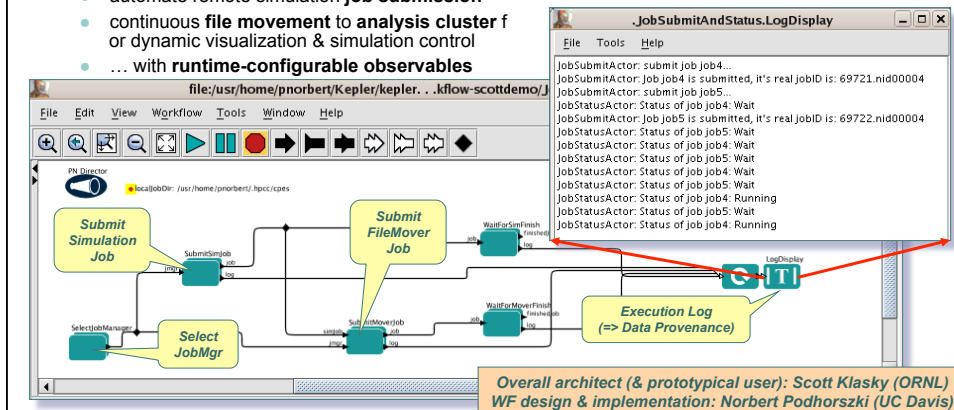
*This example shows the use of data ontologies and database wrappers.*

Service Name	Document Type	
IONB Metacat EcoGrid QueryInterface	Ecological Metadata Language 2.0.0	<input checked="" type="checkbox"/>
	Ecological Metadata Language 2.0.1	<input checked="" type="checkbox"/>
KU Digir EcoGrid QueryInterface	Darwin Core 1.0	<input checked="" type="checkbox"/>
OEON Search QueryInterface	ADEPTIDLESENASA 0.6.50	<input checked="" type="checkbox"/>

## Kepler as an Interface to the Grid

### CPES Fusion Simulation Workflow

- **Fusion Simulation Codes:** (a) GTC; (b) XGC with M3D
  - e.g. (a) currently 4,800 (soon: 9,600) nodes Cray XT3; 9.6TB RAM; 1.5TB simulation data/run
- **GOAL:**
  - automate remote simulation **job submission**
  - continuous **file movement** to analysis cluster for dynamic visualization & simulation control
  - ... with **runtime-configurable observables**



**Overall architect (& prototypical user): Scott Klasky (ORNL)**  
**WF design & implementation: Norbert Podhorszki (UC Davis)**

```

JobSubmitActor: submit job job4...
JobSubmitActor: job job4 is submitted, it's real jobID is: 69722.nid00004
JobSubmitActor: submit job job5...
JobSubmitActor: job job5 is submitted, it's real jobID is: 69722.nid00004
JobStatusActor: Status of job job4: Wait
JobStatusActor: Status of job job5: Wait
JobStatusActor: Status of job job4: Wait
JobStatusActor: Status of job job5: Wait
JobStatusActor: Status of job job4: Wait
JobStatusActor: Status of job job5: Wait
JobStatusActor: Status of job job4: Running
JobStatusActor: Status of job job5: Wait
JobStatusActor: Status of job job4: Running
    
```

## Leverage: Kepler is a Team Effort

Kepler

**Other contributors:**

- Chesire (UK Text Mining Center)
- DART (Great Barrier Reef, Australia)
- National Digital Archives + UCSD-TV (US)
- ...

Contributor names and funding info are at the Kepler website: <http://kepler-project.org>

Lee, Berkeley 108

## Graph Transformation

*Model transformation workflow specifies iterative graph rewriting to transform the top-right model into the bottom-left model.*

**DDF Director**

Model Size Reduction

Click the run button to observe the behavior reflected in the popup windows. The new window to the left shows the original model. The one to the right shows the result of static evaluation step by step.

This model demonstrates how one can possibly optimize a model. The original input is the model in BaseModel.xml, which the FileReader actor reads in. The contents of this model are then converted into an ActorToken by the ModelGenerator. OptimizesOnce is a transformation rule that gets repeatedly applied to this model until no further optimization is possible (i.e., a fixpoint is reached). In each application, two Consts that are wired to an AddSubtract actor, a MultiplyDivide actor, or a Maximum actor are replaced by a single Const with the statically computed value.

Author: Thomas Huining Feng (Inspired by Thomas Mandl)

**SDF Director**

This is the model for transformation. It has a number of Const actors connected to arithmetic operators, which can be statically evaluated. See ConstOptimization.xml.

Author: Thomas Huining Feng

Executing the model at the left transforms the top model into the bottom model.

Lee, Berkeley 109

## Workflows

```

    graph TD
      InputHostModel -- "delta: Infinity" --> InitHostModel
      InputHostModel -- "delta: Infinity" --> Transform
      Transform -- "delta: 0.8" --> InputHostModel
      Transform -- "guard: Matched, delta: 0.8" --> Finish
      Transform -- "guard: !Matched" --> OutputHostModel
      OutputHostModel -- "delta: 0.8" --> Transform
      Finish -- "delta: 0.8" --> OutputHostModel
  
```

Here we have used Event-Relationship graphs [Schruben 83] to specify the workflow logic.

### Model Size Reduction

Click the run to execute. The new window to the left shows the original model. The right one shows the result of static evaluation step by step.

As how one can original input is the il, which the contents of this model an ActorToken by the ModelGenerator. formation rule that gets repeatedly until no further optimization is possible. In each application, two Consts that tract actor, a MultiplyDivide actor, or a Maximum single Const with the statically computed value.

xml in the same directory contains a DDF model that has exactly but has a much more complicated design for not using an ERG ler.

file:/ptii/ptolemy/actor/gtd/demo/Cons...ransformationController.Transformation

- Constraint: A.input.getWidth() == 2
- Constraint2: A.getClassName().equals("ptolemy.actor.lib.AddSubtract") && A.input.getName().equals("plus") || A.getClassName().equals("ptolemy.actor.lib.MultiplyDivide") && A.input.getName().equals("multiply") && A.getClassName().equals("ptolemy.actor.lib.Maximum")
- Constraint3: A.output.getWidth() == 1

## Some Current Research Thrusts in the Ptolemy Project

- Precision-timed (PRET) machines:** Introduce timing into the core abstractions of computing, beginning with instruction set architectures, using configurable hardware as an experimental platform.
- Distributed real-time computing (PTIDES):** Models of computation based on distributed discrete events, embedded OS (PtidyOS), analysis and synthesis techniques.
- Model engineering:** Modeling and design of large scale systems, those that include networking, database, grid computing, and information subsystems.
- Semantics of concurrent and real-time systems:** Mathematical models of programs in conjunction with models of their physical environment.

Lee, Berkeley 111



## Forthcoming Book

### Chapters

1. Heterogeneous Modeling
2. Building Graphical Models
3. Dataflow
4. Process Networks and Rendezvous
5. Synchronous/Reactive Models
6. Finite State Machines
7. Discrete Event Models
8. Modal Models
9. Continuous Time Models
10. Cyber-Physical Systems

### Appendices

- A. Expressions
- B. Signal Display
- C. The Type System
- D. Creating Web Pages

# System Design, Modeling, and Simulation

Claudius Ptolemaeus, Editor,  
UC Berkeley

<http://Ptolemy.org>