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.

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

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*

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/

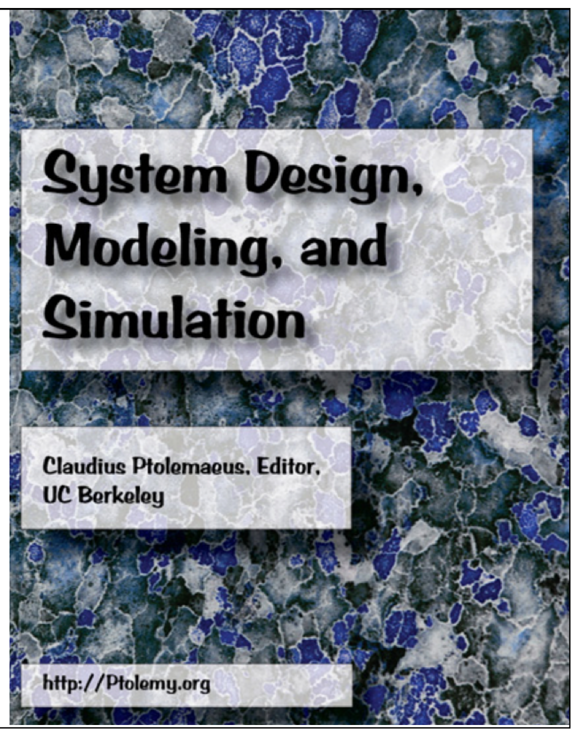
Forthcoming Book

Chapters

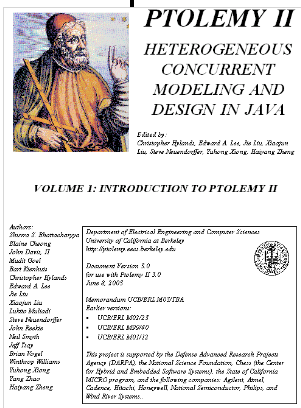
1. Heterogeneous Modeling
2. Building Graphical Models
3. Dataflow
4. Process Networks and Rendezvous
5. Synchronous/Reactive Models
6. Finite State Machines
7. Discrete Event Models
8. Modal Models
9. Continuous Time Models
10. Cyber-Physical Systems

Appendices

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



Getting More Information: Documentation



Volume 1:
User-Oriented

Volume 2:
Developer-Oriented

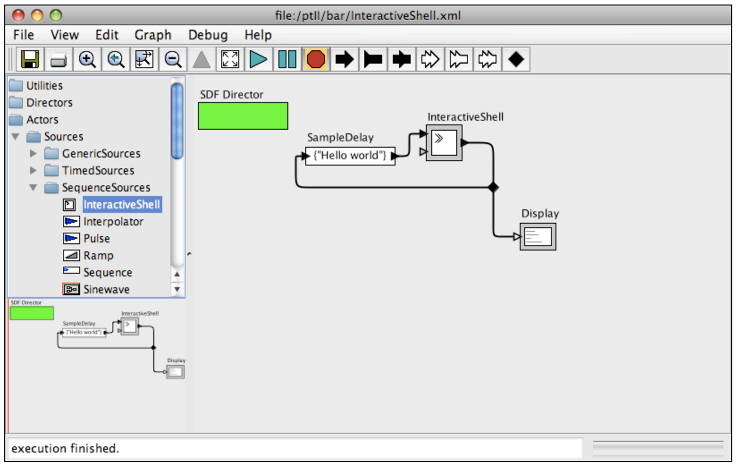
Volume 3:
Researcher-Oriented

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

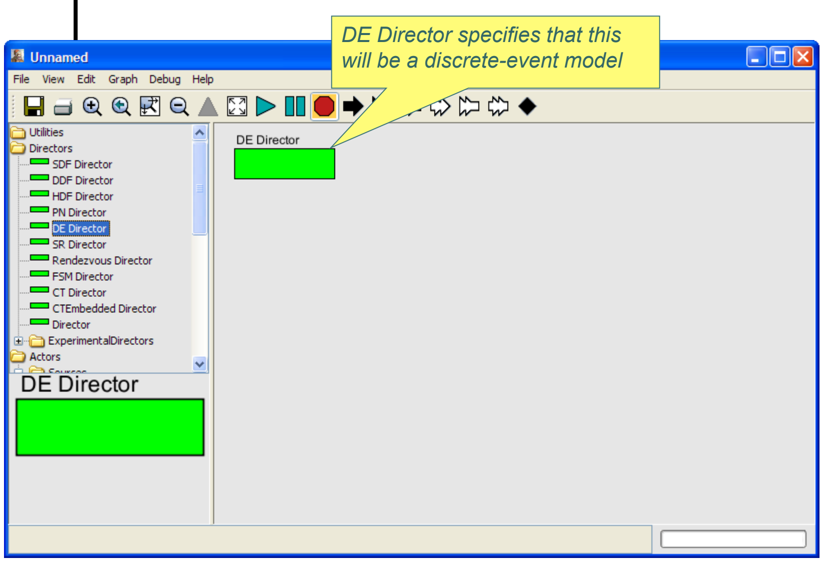
Outline

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

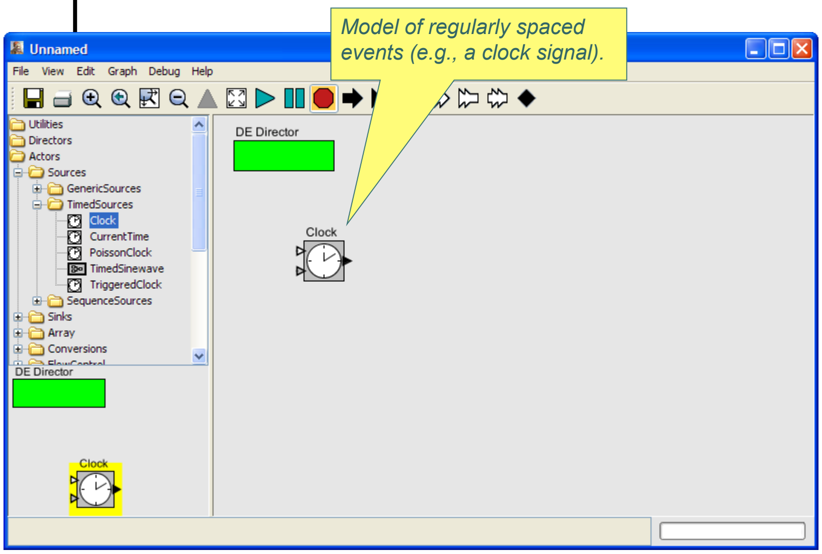
Building Models – Hello World



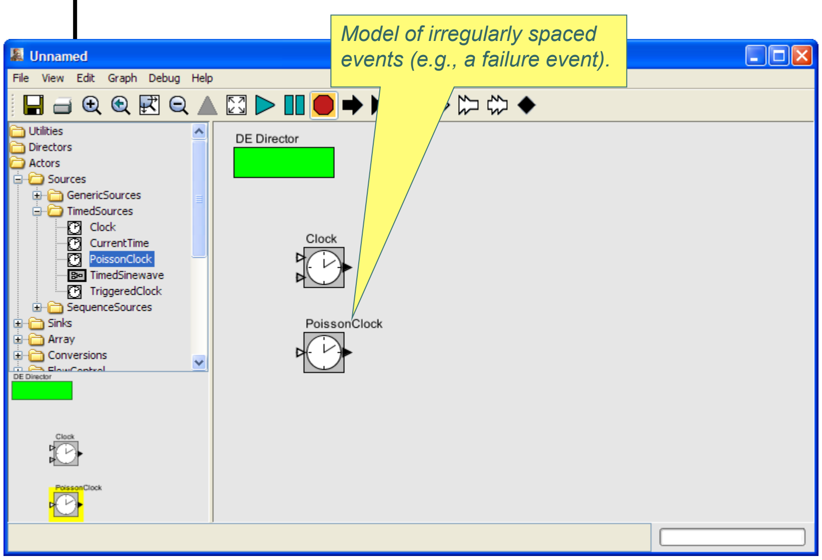
Building more interesting models



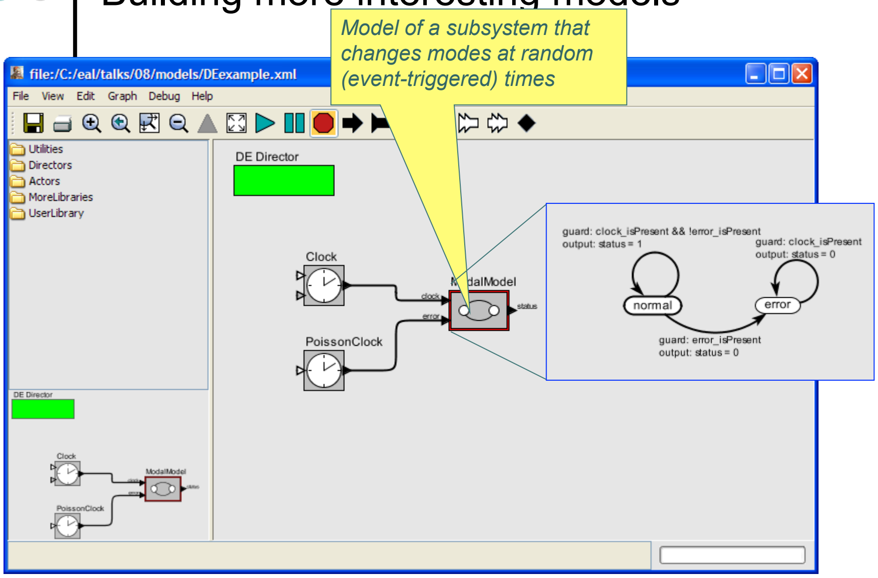
Building more interesting models



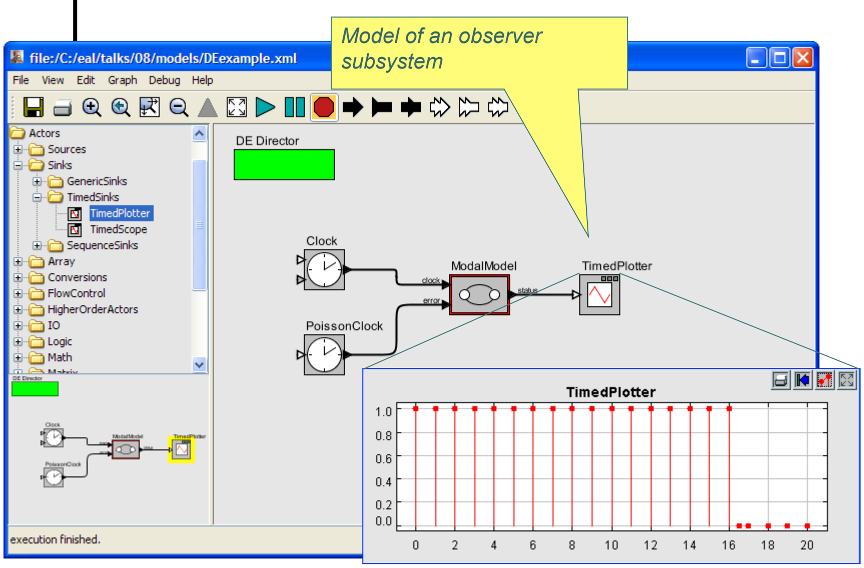
Building more interesting models



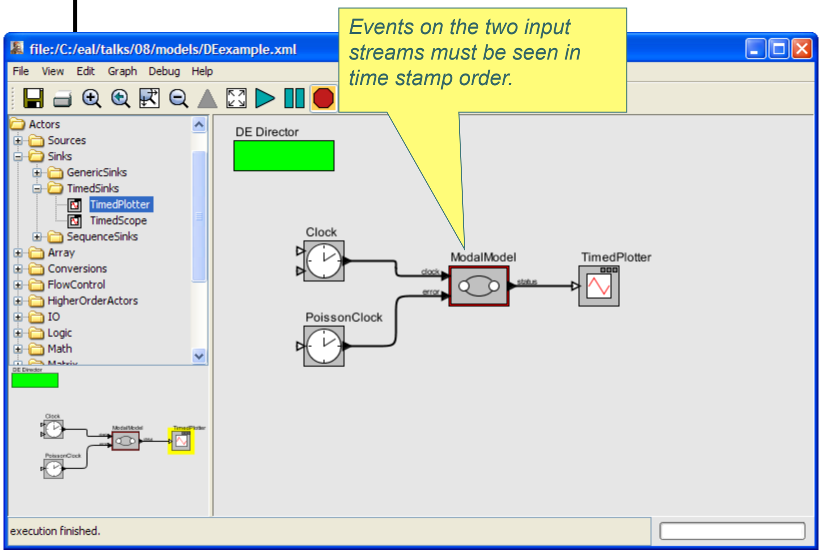
Building more interesting models



Building more interesting models



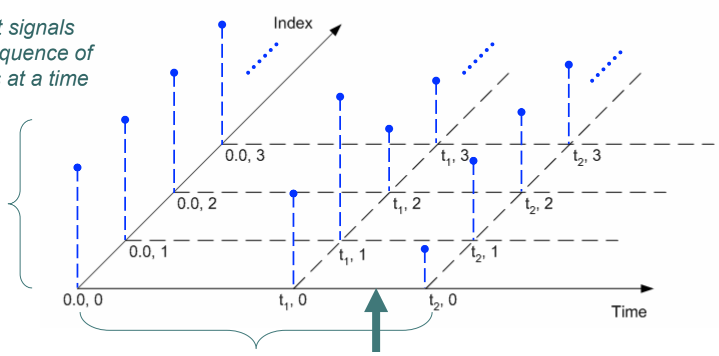
Building more interesting models



Ptolemy uses *Superdense* Time

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

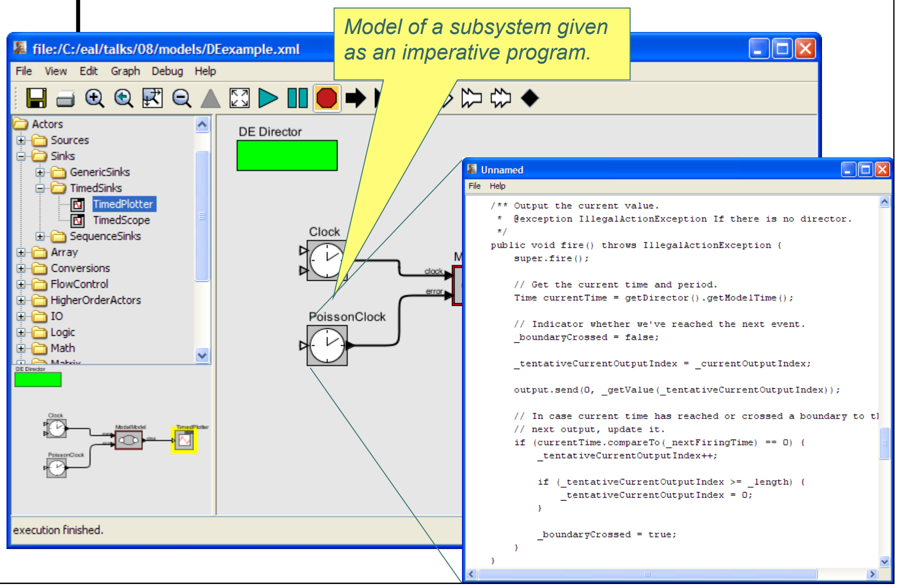
Values V



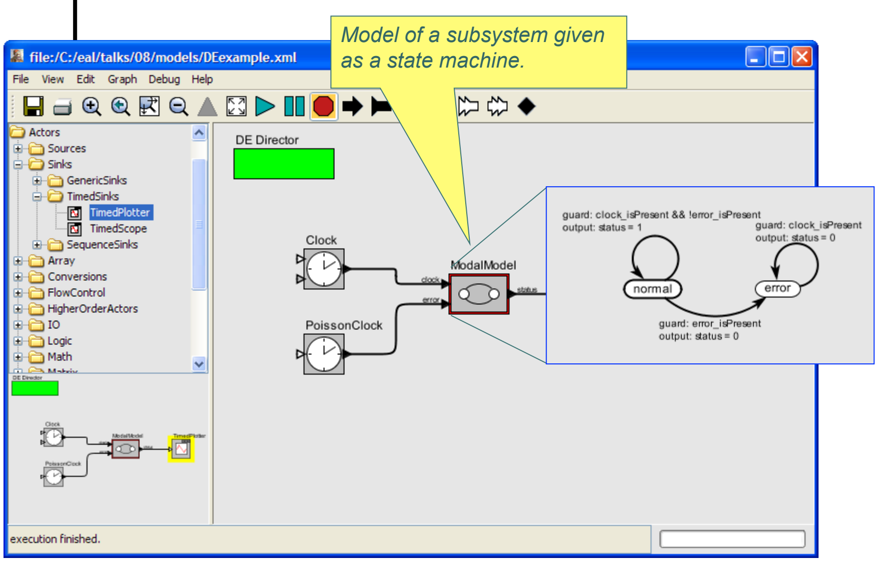
Initial segment $I \subseteq \mathbb{R}_+ \times \mathbb{N}$ where the signal is defined

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

This is a Component Technology

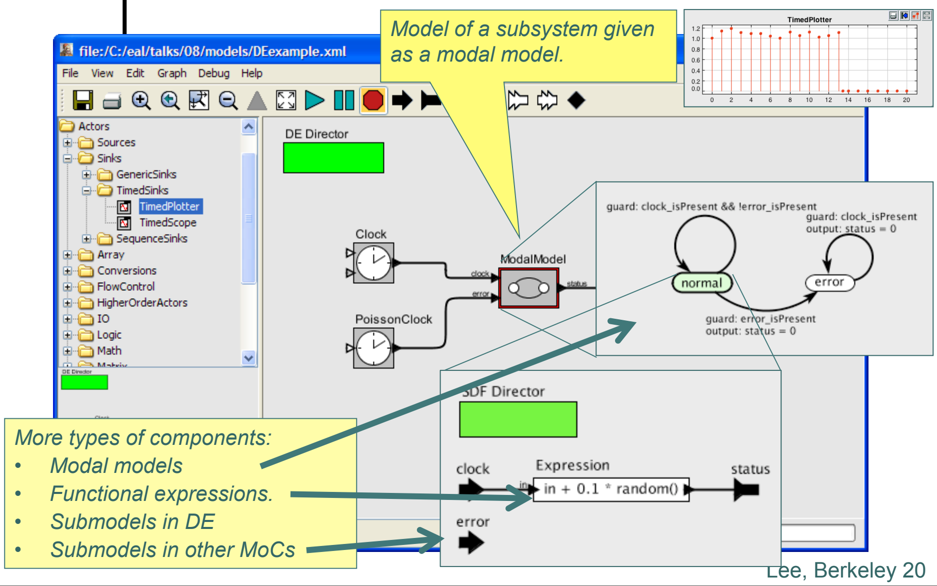


This is a Component Technology

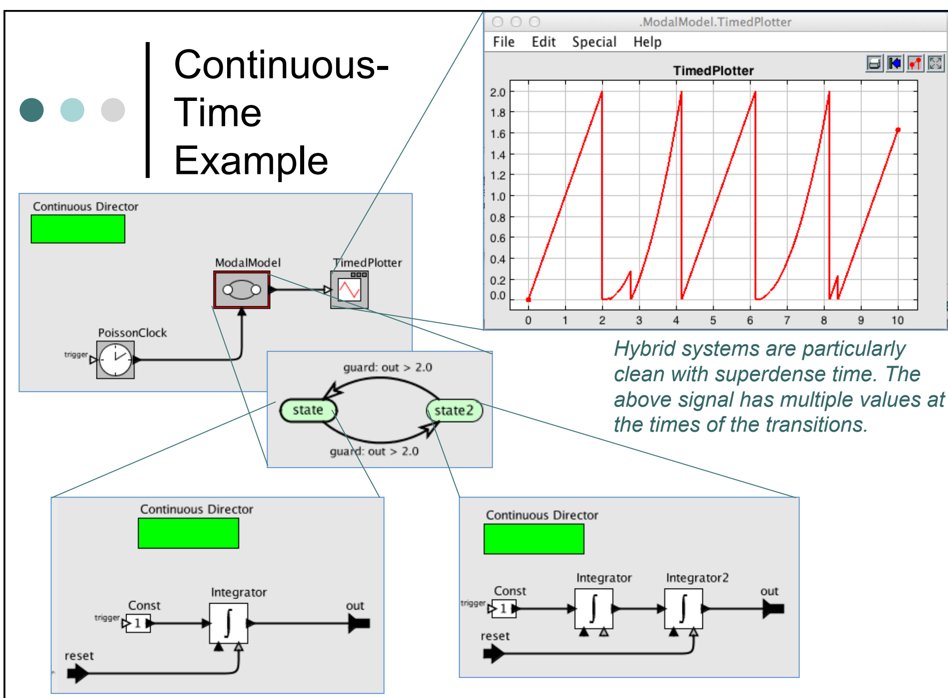


Lee, Berkeley 19

This is a Component Technology

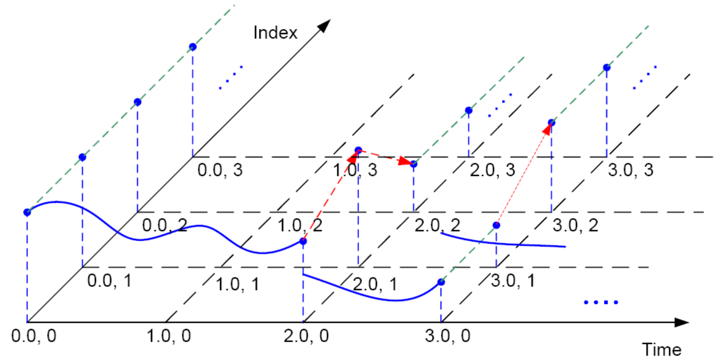


Continuous-Time Example



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

Superdense Time for Continuous-Time Signals

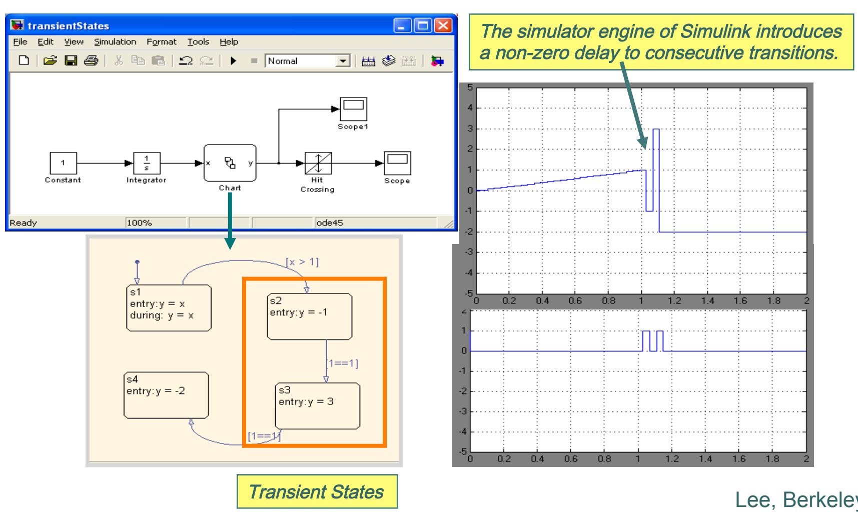


At each tag, the signal has exactly one value. At each time point, the signal has an infinite number of values. The red arrows indicate value changes between tags, which correspond to discontinuities. Signals are piecewise continuous, in a well-defined technical sense.

22

Contrast with Simulink/Stateflow

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

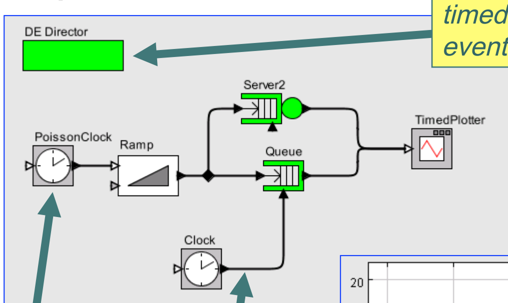


Outline

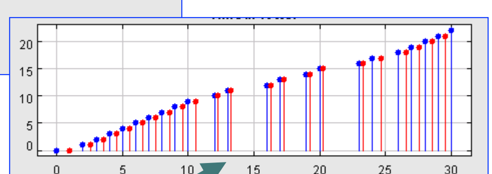
- Building models
- **Models of computation (MoCs)**
- Creating actors
- Creating directors
- Software architecture
- Miscellaneous topics

MoC Example 1: Discrete Events (DE)

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.



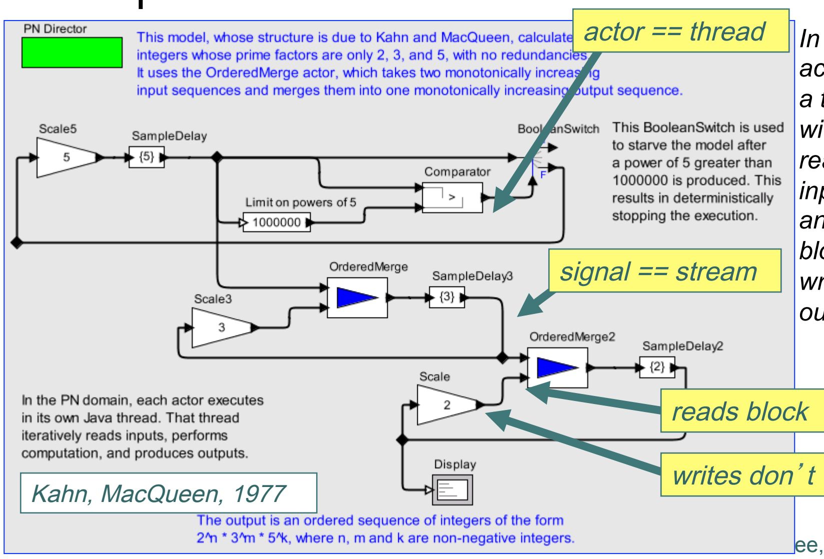
Event source

Signal

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

Time line

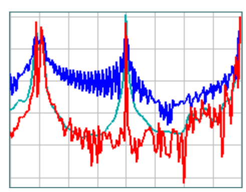
MoC Example 2:
Kahn Process Networks (PN)

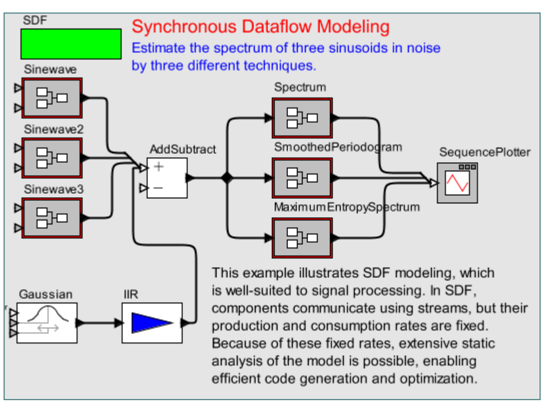


In PN, every actor runs in a thread, with blocking reads of input ports and non-blocking writes to outputs.

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.

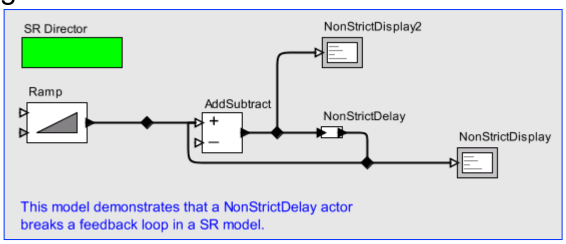


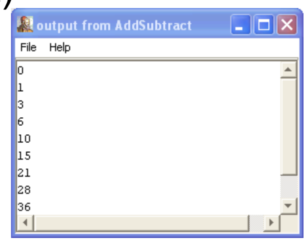


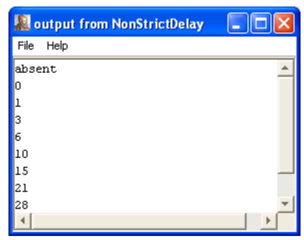
SDF is a special case of PN where deadlock and boundedness are decidable. It is well suited to static scheduling and code generation. It can also be automatically parallelized.

MoC Example 4: Synchronous/Reactive (SR)

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



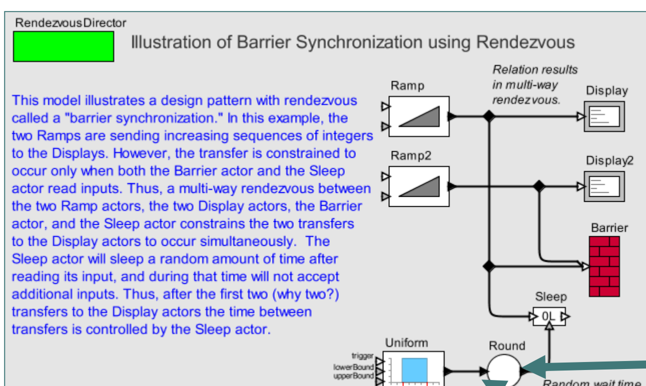




Like SDF, SR is decidable and suitable for code generation. It is harder to parallelize than SDF, however.

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

MoC Example 5: Rendezvous



writes block

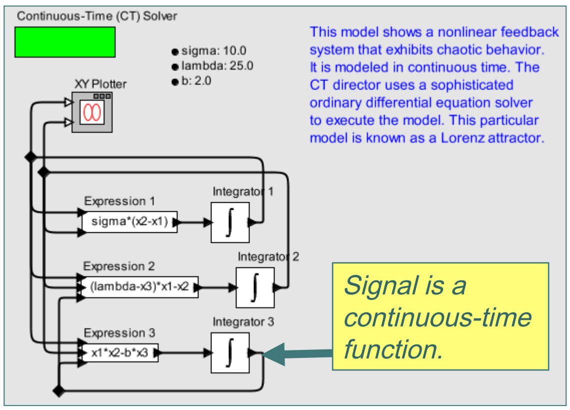
In Rendezvous, every actor runs in a thread, with blocking reads of input ports and blocking writes to outputs. Every communication is a (possibly multi-way) rendezvous.

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

actor == thread

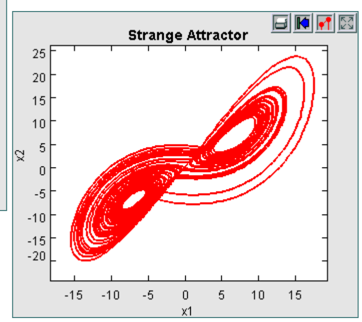
reads block

MoC Example 6: Continuous Time (CT)



Director includes an ODE solver.

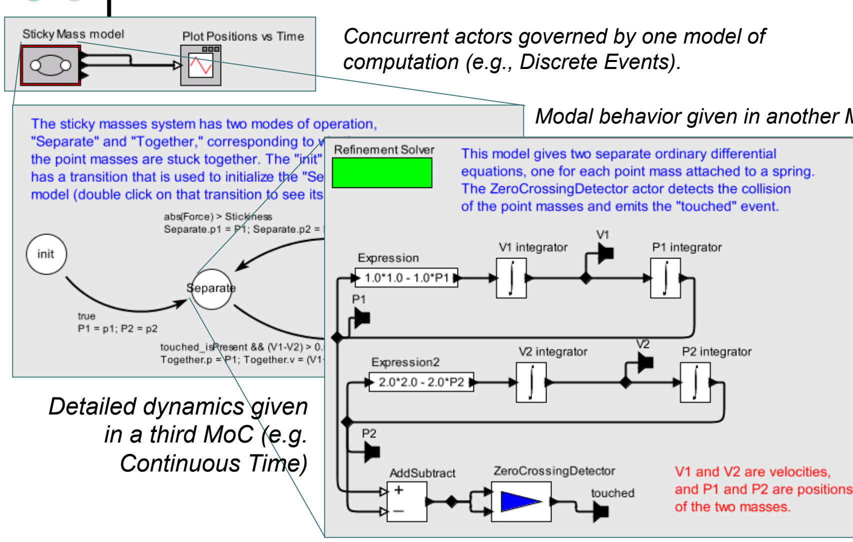
In CT, actors operate on continuous-time and/or discrete-event signals. An ODE solver governs the execution.



Ptolemy II Hierarchy Supports Heterogeneity

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

Modal behavior given in another MoC.



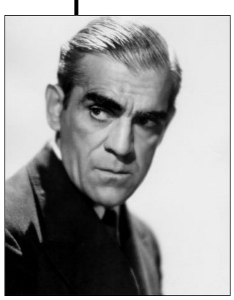
Detailed dynamics given in a third MoC (e.g. Continuous Time)

This requires a composable abstract semantics.

Outline

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

Actors






Ptolemy Components are Actors and Objects

The established: Object-oriented:

class name
data
methods

call → *return*

What flows through an object is sequential control

Things happen to objects

The alternative: Actor oriented:

Actors make things happen

actor name
data (state)
parameters
ports

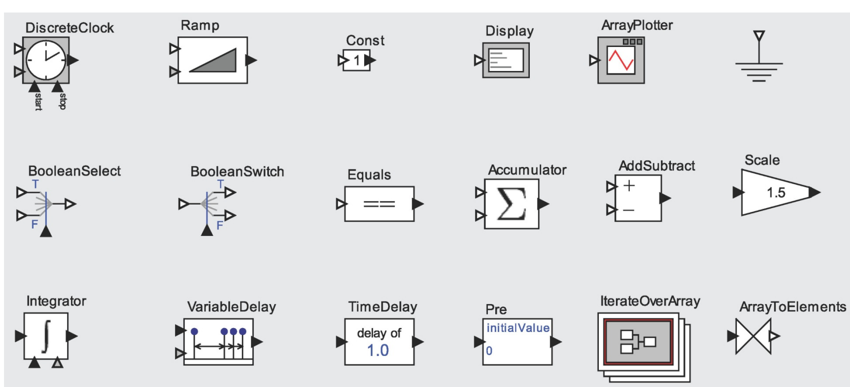
Input data → *Output data*

What flows through an object is evolving data

Lee, Berkeley 36

Actors

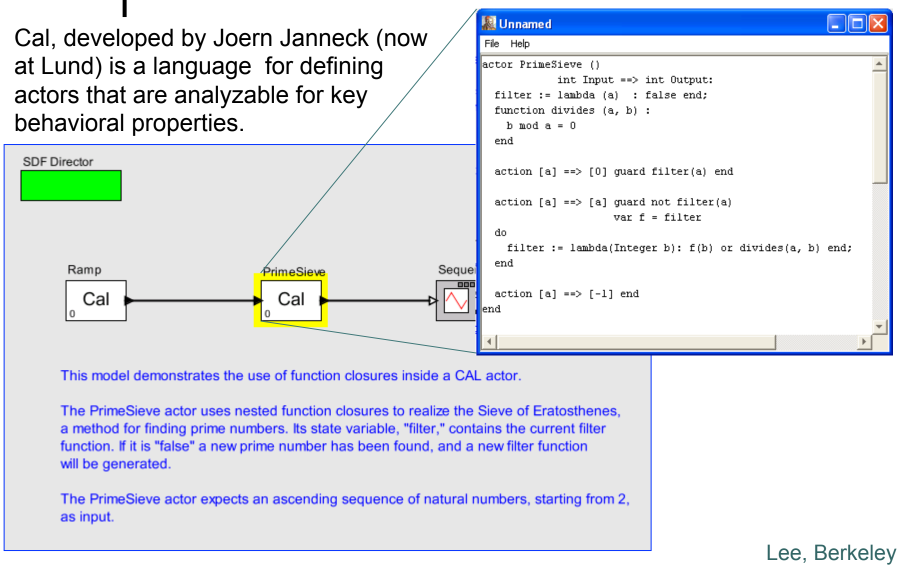
- Ptolemy has a library of predefined actors
- Java classes that implement the "executable" interface



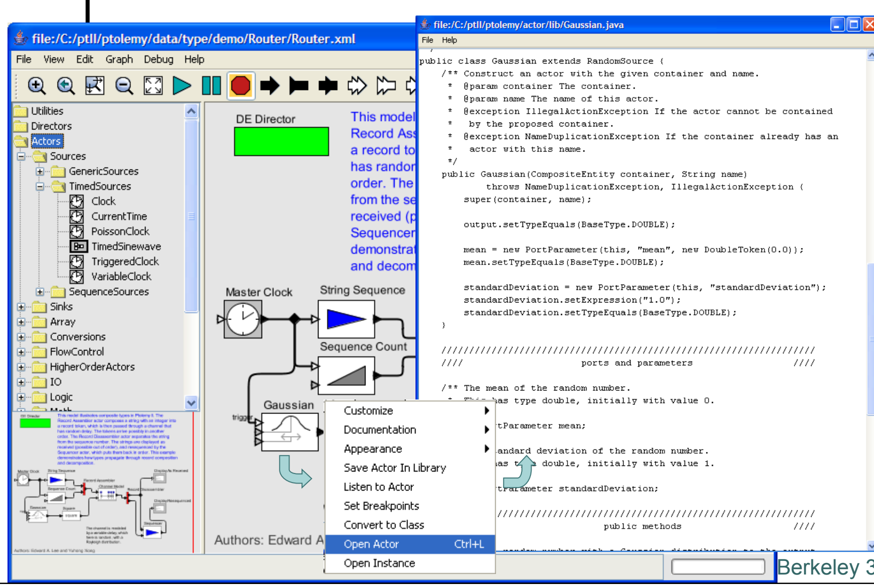
Lee, Berkeley 37

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.



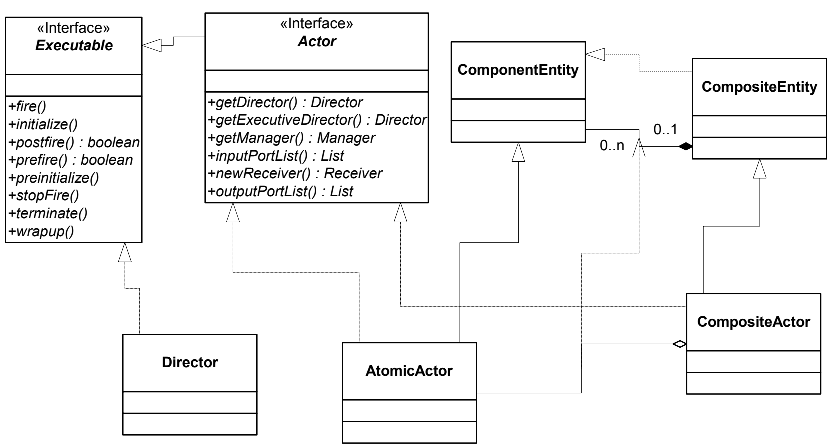
Most Actors are Written in Java



Simple String Manipulation Actor in Java

```
public class Ptolemnizer extends TypedAtomicActor {
    public Ptolemnizer(CompositeEntity container, String name)
            throws IllegalActionException, 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 IllegalActionException {
        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));
        }
    }
}
```


Object Model for Executable Components



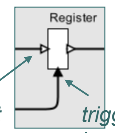
Definition of the Register Actor (Sketch)

```
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

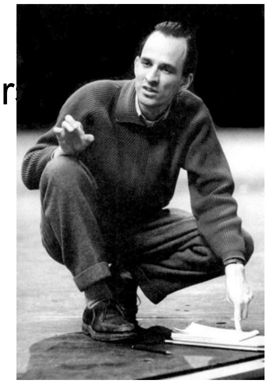
trigger input port

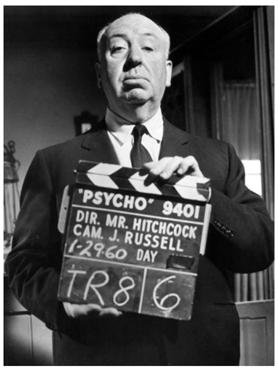
Lee, Berkeley 42

Outline

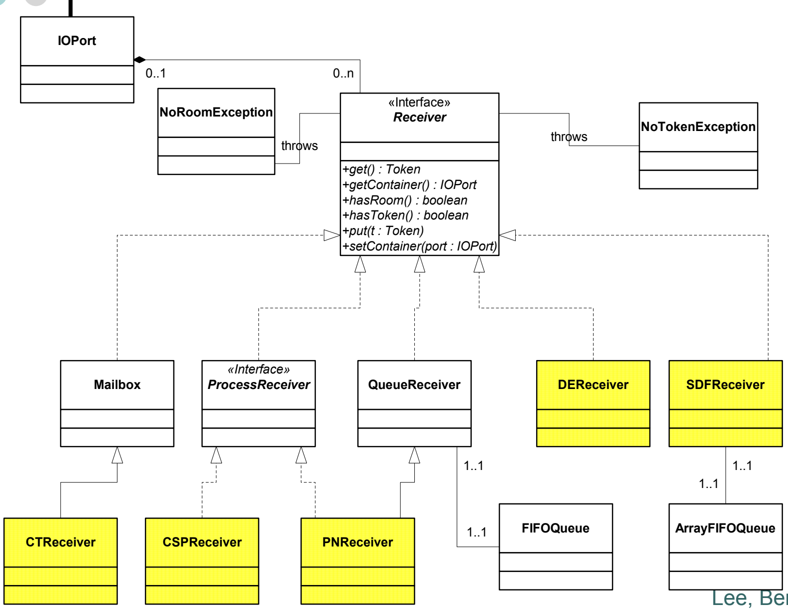
- Building models
- Models of computation (MoCs)
- Creating actors
- **Creating directors**
- Software architecture
- Miscellaneous topics

Directors





Object Model (Simplified) for Communication Infrastructure

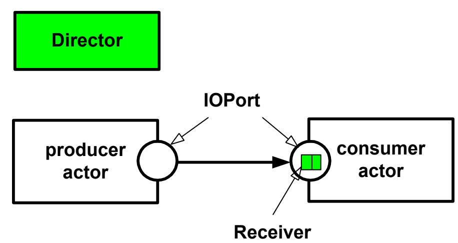


Object-Oriented Approach to Achieving Behavioral Polymorphism

«Interface»
Receiver

+get() : Token
+getContainer() : IOPort
+hasRoom() : boolean
+hasToken() : boolean
+put(t : Token)
+setContainer(port : IOPort)

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.*

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.

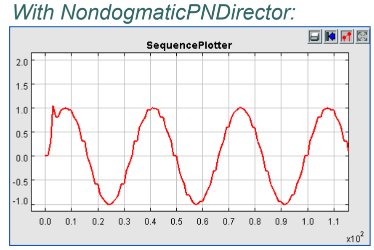
Parameterizing the behavior of a receiver is a simple form of communication refinement, a key principle in, for example, Metropolis.

Implementation of the NondogmaticPNDirector

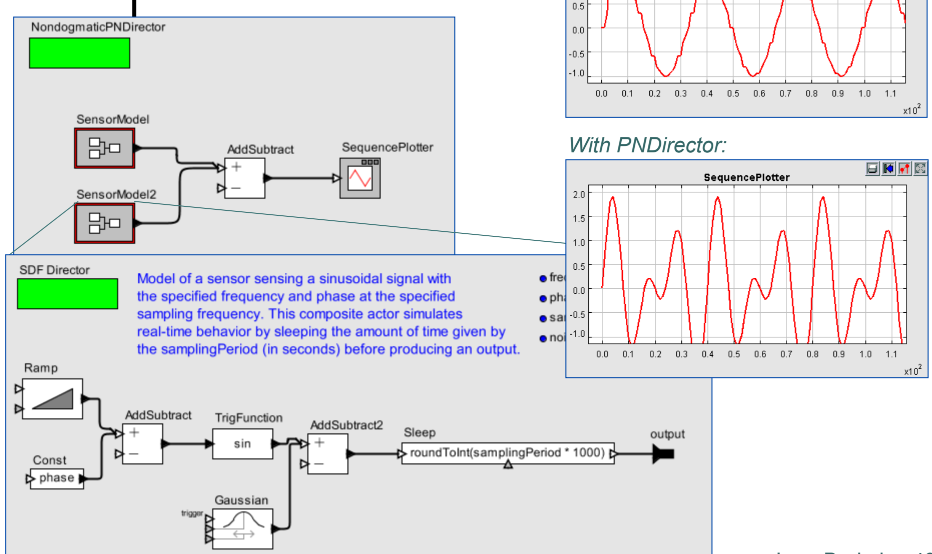
```
package doc.tutorial;
import …
public class NondogmaticPNDirector extends PNDirector {
    public NondogmaticPNDirector(CompositeEntity container, String name)
            throws IllegalActionException, NameDuplicationException {
        super(container, name);
    }
    public Receiver newReceiver() {
        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;
        }
    }
}
```


Using It

With NondogmaticPNDirector:



With PNDirector:



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.

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, …)...

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.

A few variants of dataflow MoCs

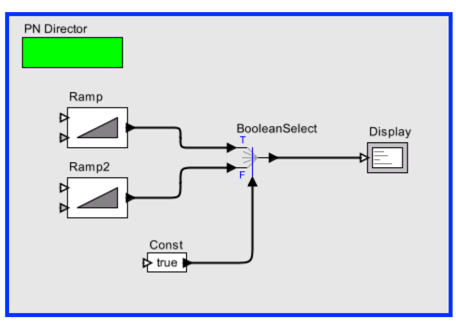
- *Computation graphs [Karp and Miller, 1966]*
- *Static dataflow [Dennis, 1974]*
- *Dynamic dataflow [Arvind, 1981]*
- *Structured dataflow [Matwin & Pietrzykowski 1985]*
- *K-bounded loops [Culler, 1986]*
- *Synchronous dataflow [Lee & Messerschmitt, 1986]*
- *Structured dataflow and LabVIEW [Kodosky, 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]*
- …

Some Subtleties

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

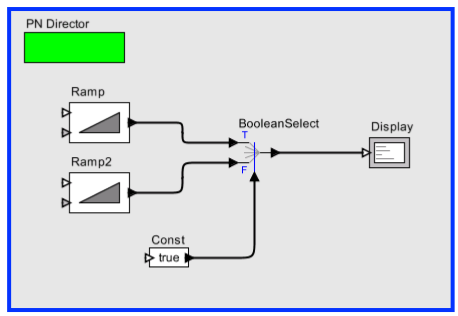
Question 1:
Is “Fair” Scheduling a Good Idea?

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



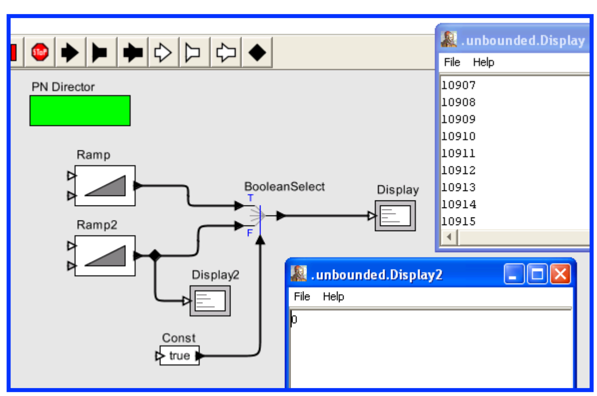
Question 2:
Is “Data-Driven” Execution a Good Idea?

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



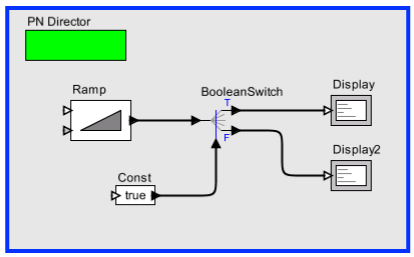
Question 3:
When are Outputs Required?

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

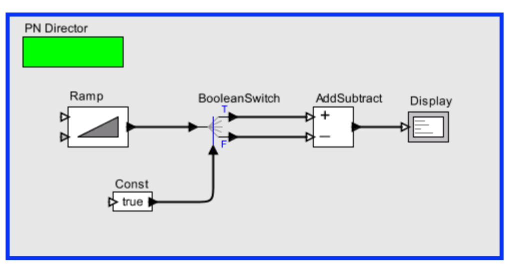


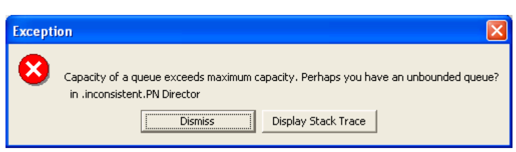
Question 4: Is “Demand-Driven” Execution a Good Idea?

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

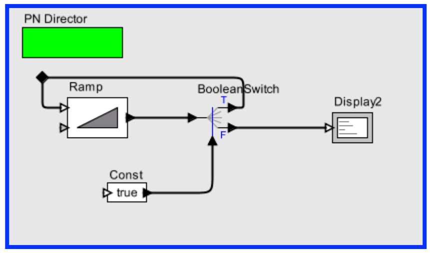


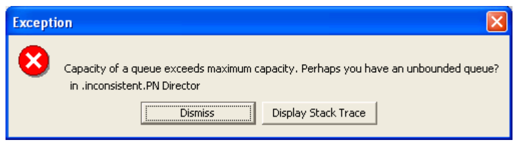
Question 5: What is the “Correct” Execution of This Program?





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





Naïve Schedulers Fail

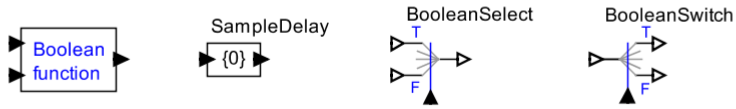
- Fair
- Demand driven
- Data driven
- Most mixtures of demand and data driven

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?

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

Undecidability and Turing Completeness [Buck 93]

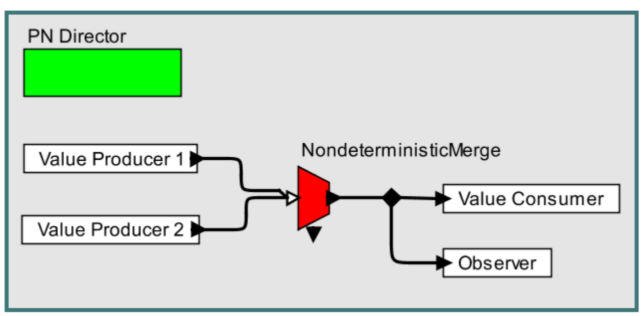
Given the following four actors and Boolean streams, you can construct a universal Turing machine:



Hence, the following questions are undecidable:

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

Question 7:
How to support nondeterminism?

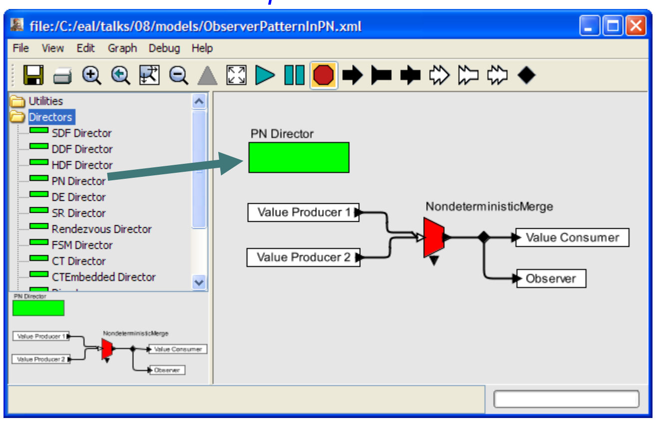


Merging of streams is needed for some applications. Does this require fairness? What does fairness mean?

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.

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.

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).

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.

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

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.

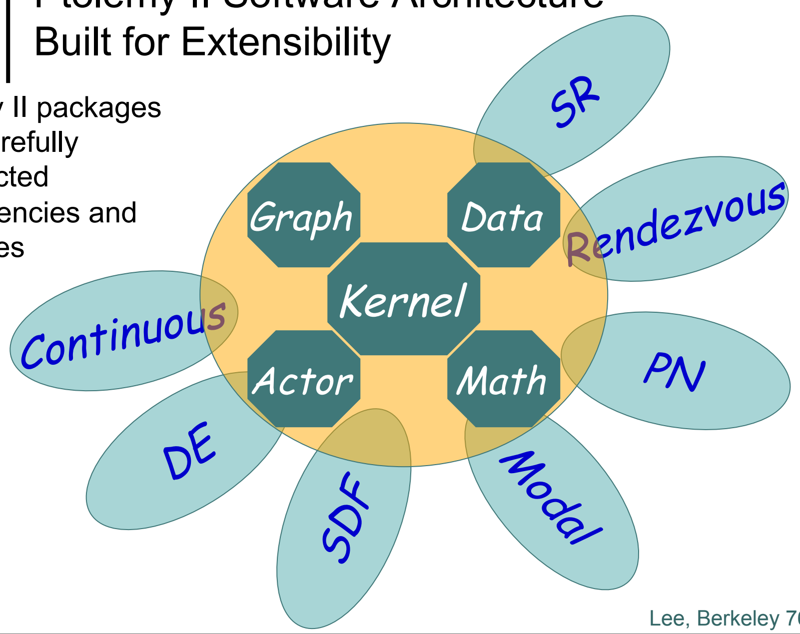
See $PTII/doc/tutorial/domains

Outline

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

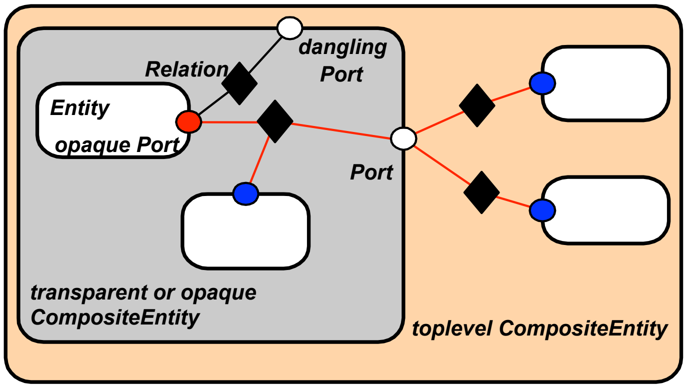
Ptolemy II Software Architecture Built for Extensibility

Ptolemy II packages have carefully constructed dependencies and interfaces



Lee, Berkeley 76

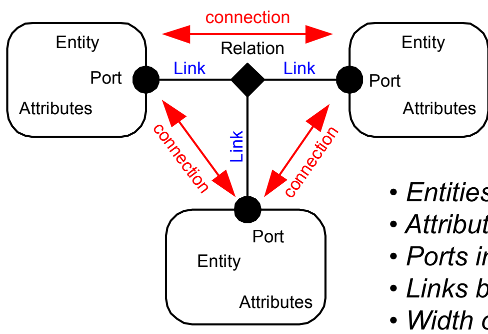
Hierarchy - Composite Components



Separable Tool Architecture

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

The Basic Abstract Syntax for Composition



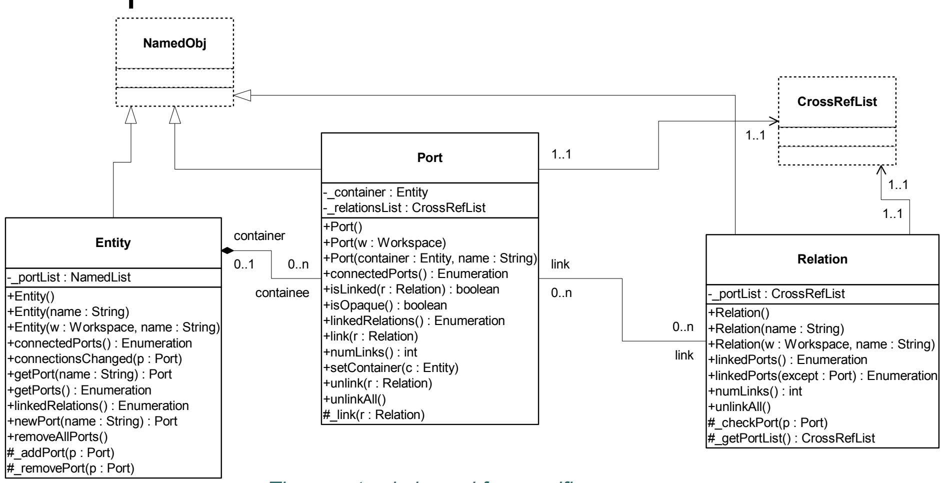
- *Entities*
- *Attributes on entities (parameters)*
- *Ports in entities*
- *Links between ports*
- *Width on links (channels)*
- *Hierarchy*

Concrete syntaxes:

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

Lee, Berkeley 79

Meta Model: Kernel Classes Supporting the Abstract Syntax



These get subclassed for specific purposes.

Lee, Berkeley 80

Separable Tool Archictecture

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

MoML
XML Schema for this Abstract Syntax

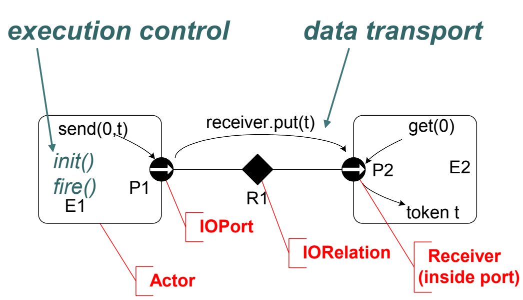
Ptolemy II designs are represented in XML:

```
...
<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)*
- *…*

Implemented as a Java interface

Interface "Executable"

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

Example execution sequence

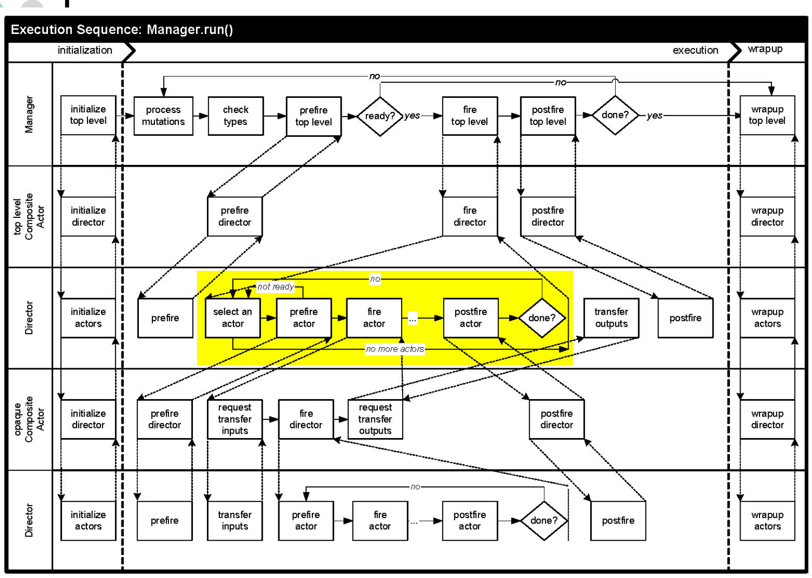
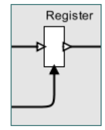


FIGURE 2.14. Example execution sequence implemented by run() method of the Director class.

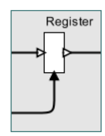
How Does This Work? Execution of Ptolemy II Actors



Flow of control:

- Initialization
- Execution
- Finalization

How Does This Work? Execution of Ptolemy II Actors

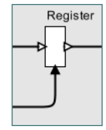


Flow of control:

- Initialization
- Execution
- Finalization

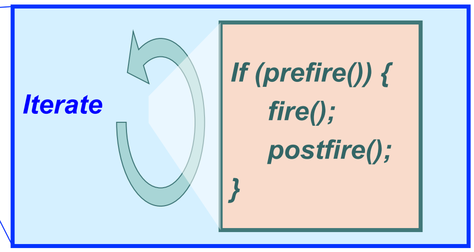
E.g., in DE: Post tags on the event queue corresponding to any initial events the actor wants to produce.

How Does This Work? Execution of Ptolemy II Actors



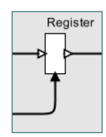
Flow of control:

- Initialization
- Execution
- Finalization



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

How Does This Work? Execution of Ptolemy II Actors



Flow of control:

- Initialization
- Execution
- Finalization

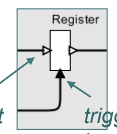
Definition of the Register Actor (Sketch)

```
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?

React to trigger input.

Read the data input and update the state.



data input port

trigger input port

Lee, Berkeley 91

Separable Tool Archictecture

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

Lee, Berkeley 92

Models of Computation Implemented in Ptolemy II

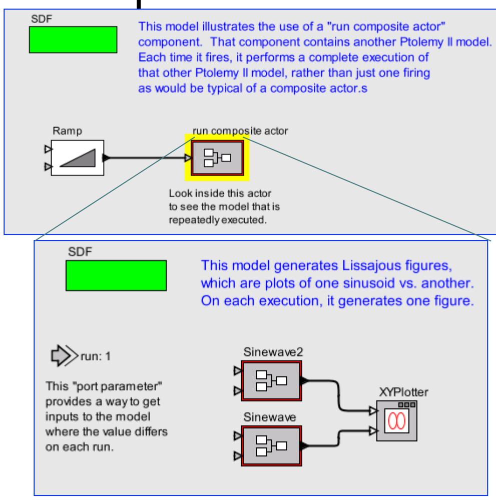
- CI – Push/pull component interaction
- Click – Push/pull with method invocation
- CSP – concurrent threads with rendezvous
- Continuous – continuous-time modeling with fixed-point semantics
- CT – continuous-time modeling
- DDF – Dynamic dataflow
- DE – discrete-event systems
- DDE – distributed discrete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

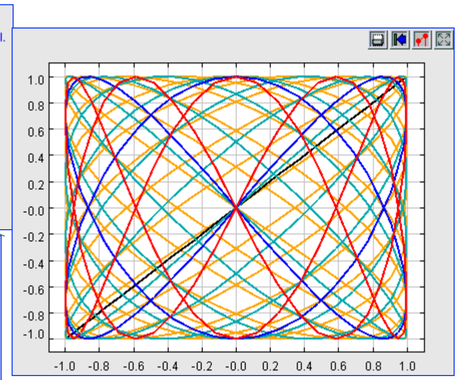
Most of these are actor oriented.

Outline

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

Example Extensions
Using Models to Control Models

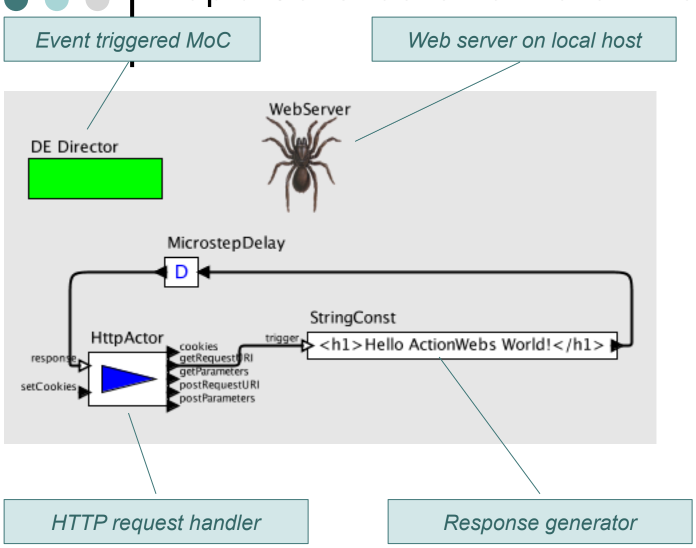




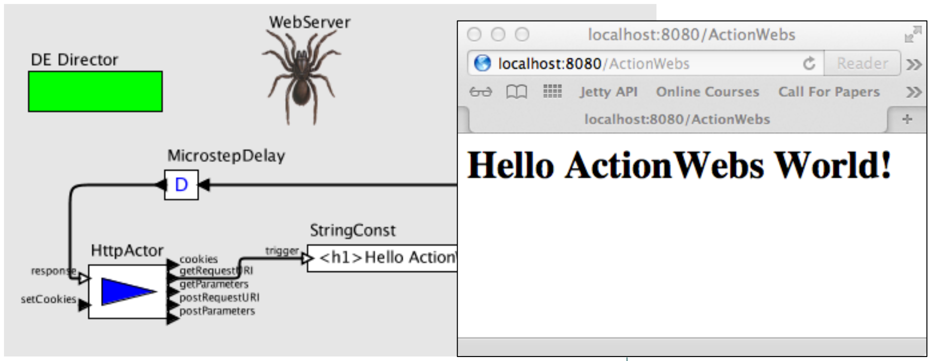
This is an example of a “higher-order component,” or an actor that references one or more other actors.

MultipleRuns SDF demo

Example Extensions:
Rapid Construction of Action Web Services



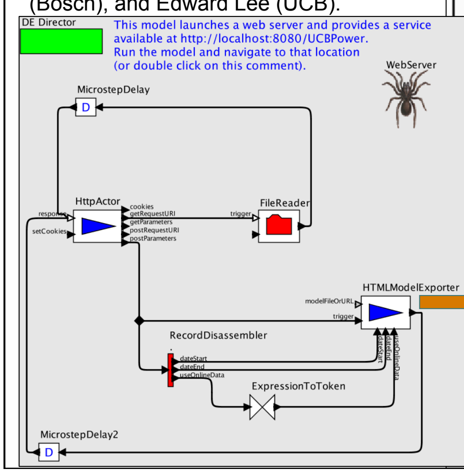
Example Extensions: Rapid Construction of Action Web Services

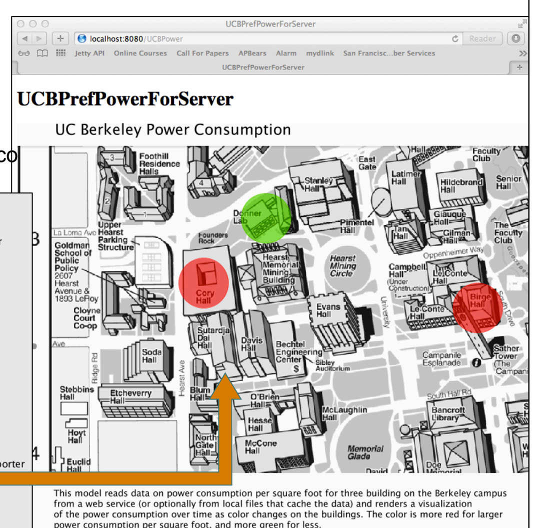


Response to an HTTP get request

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).

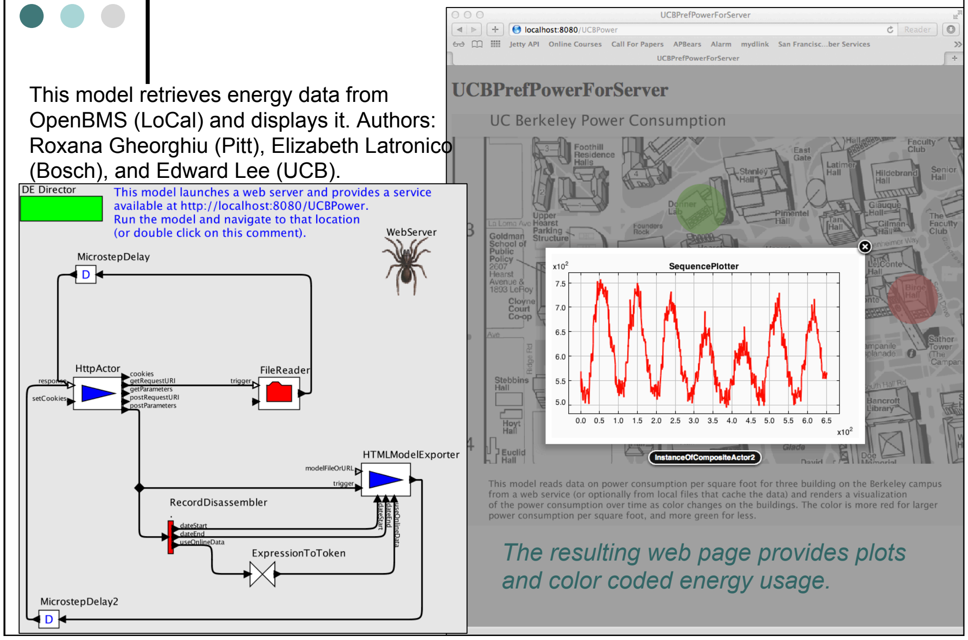




HTTP post specifies parameter for a Ptolemy model that runs on the server and then exports itself to a web page.

Example: UCB Power

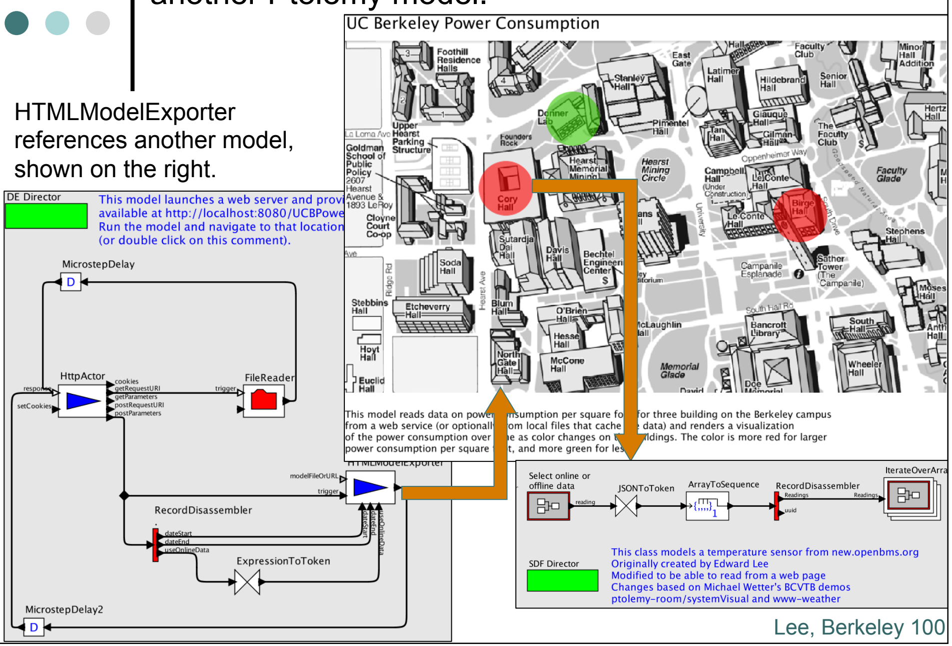
This model retrieves energy data from OpenBMS (LoCal) and displays it. Authors: Roxana Gheorghiu (Pitt), Elizabeth Latronico (Bosch), and Edward Lee (UCB).



The resulting web page provides plots and color coded energy usage.

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

HTMLModelExporter references another model, shown on the right.



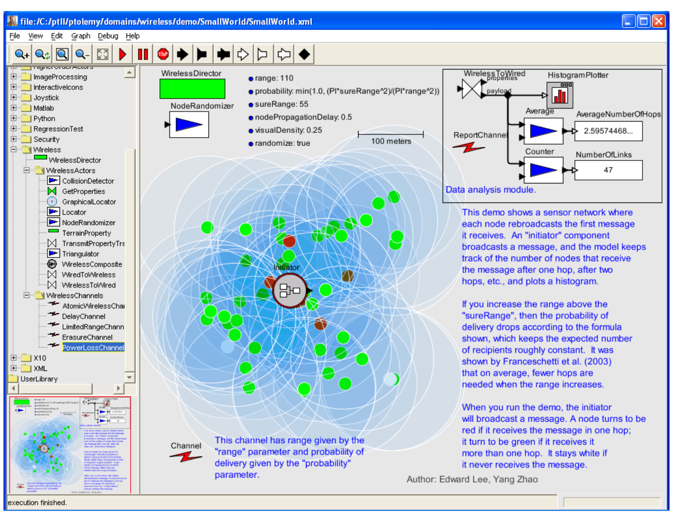
Ptolemy II Extension Points

- 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.

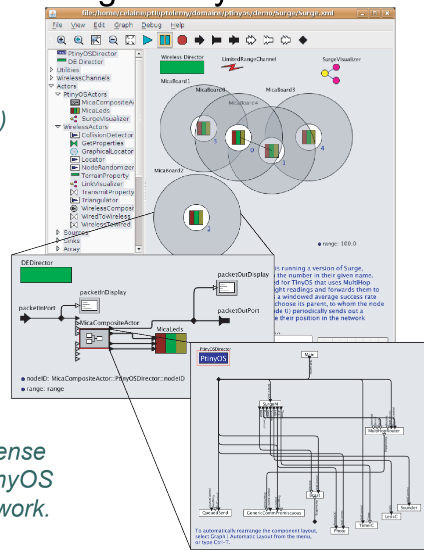
The example at the left shows a grid of nodes that relay messages from an initiator (center) via a channel that models a low (but non-zero) probability of long range links being viable.

Lee, Berkeley 103

Viptos: Extension of VisualSense with Programming of TinyOS nodes

Viptos demo:
Multihop routing (Surge)

Hardware



Physical environment

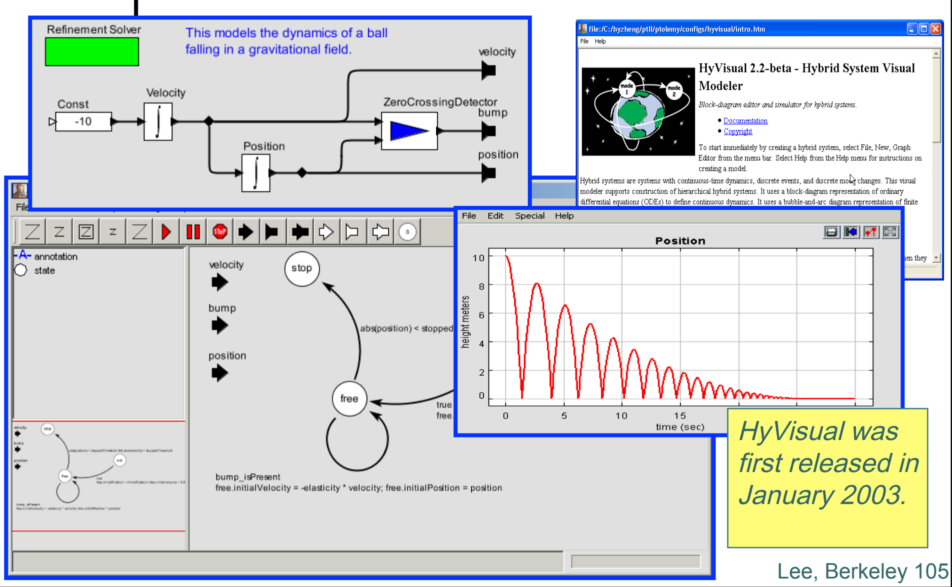
Simulation (with visualization of routing tree)

Software

Code generation: Models to nesC.

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

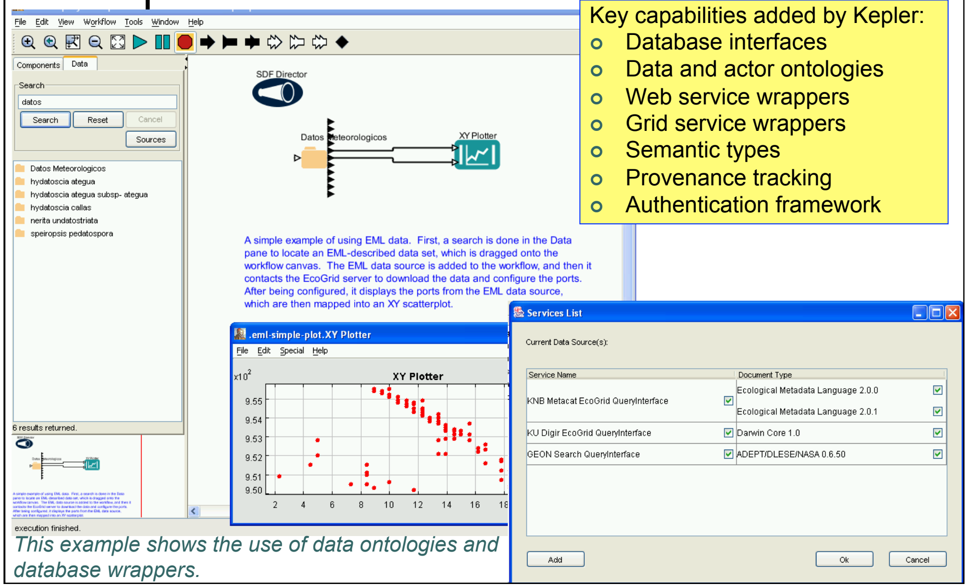
Another Extension: HyVisual – Hybrid System Modeling Tool Based on Ptolemy II



HyVisual was first released in January 2003.

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

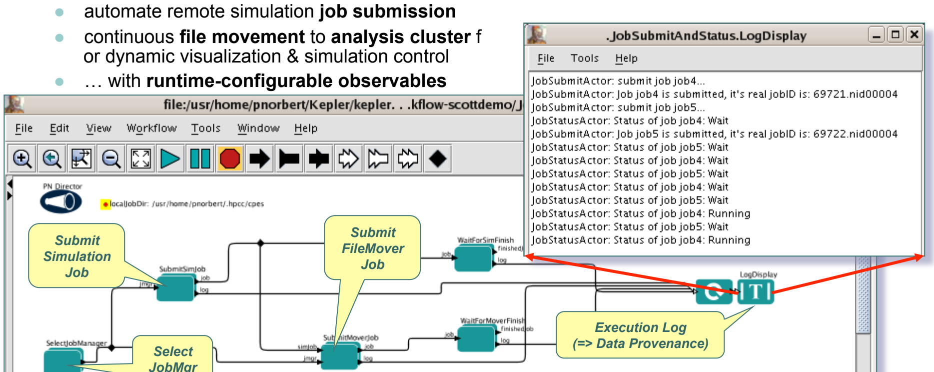


This example shows the use of data ontologies and database wrappers.

Kepler as an Interface to the Grid

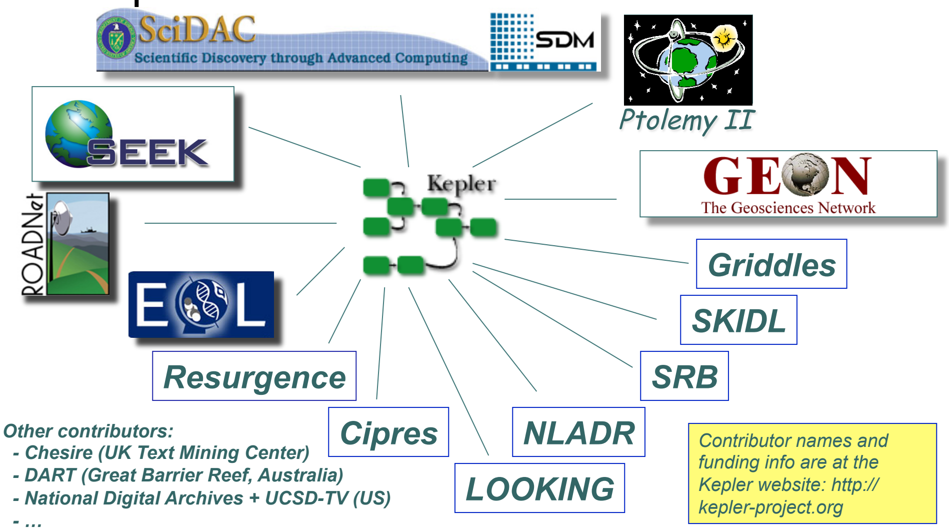
CPES Fusion Simulation Workflow

- **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)

Leverage: Kepler is a Team Effort



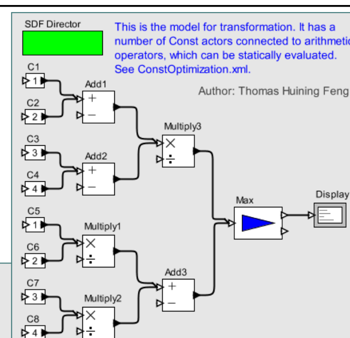
Other contributors:
- *Chesire (UK Text Mining Center)*
- *DART (Great Barrier Reef, Australia)*
- *National Digital Archives + UCSD-TV (US)*
- *...*

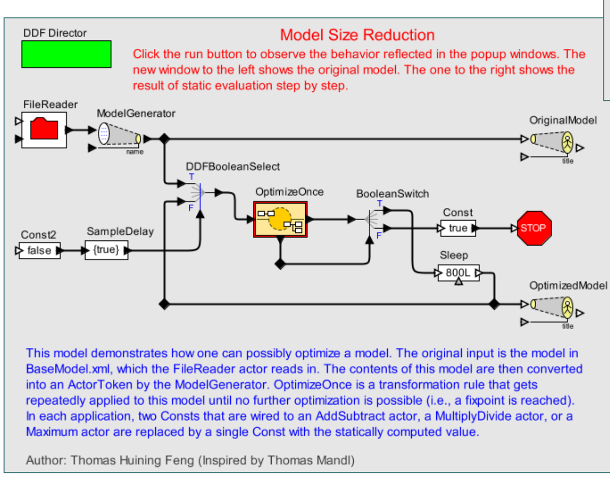
Contributor names and funding info are at the Kepler website: http://kepler-project.org

Lee, Berkeley 108

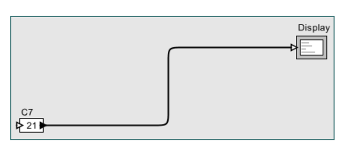
Graph Transformation

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



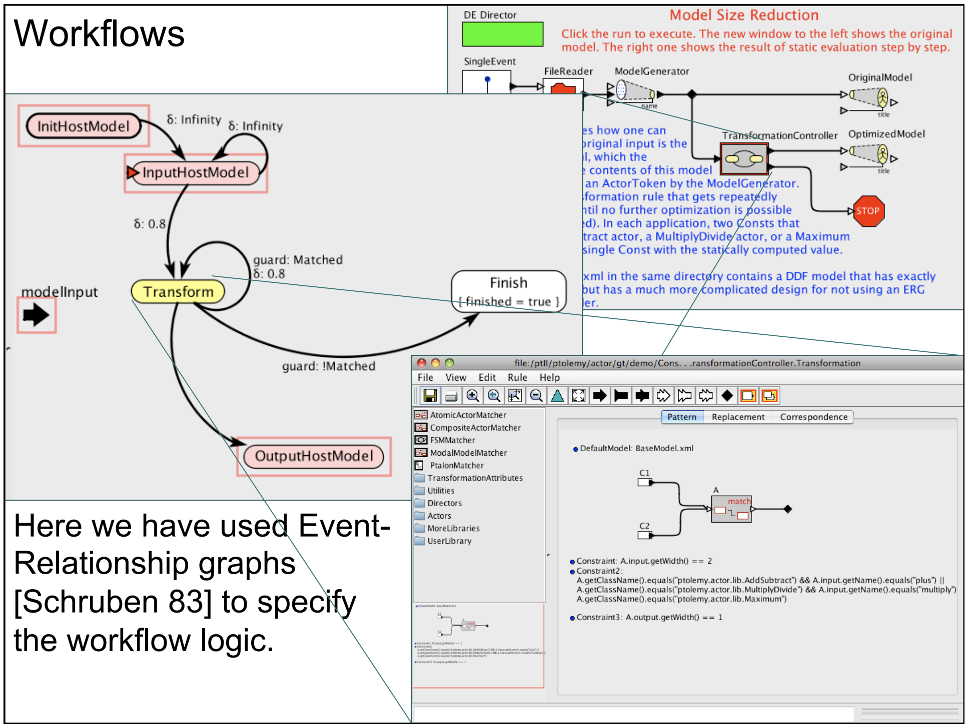


Executing the model at the left transforms the top model into the bottom model.



Lee, Berkeley 109

Workflows



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

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.

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