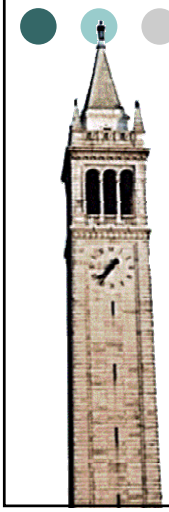


Concurrent Models of Computation

Edward A. Lee

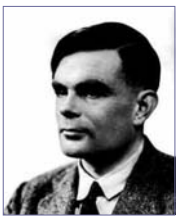
Professor, UC Berkeley

Ptolemy Project

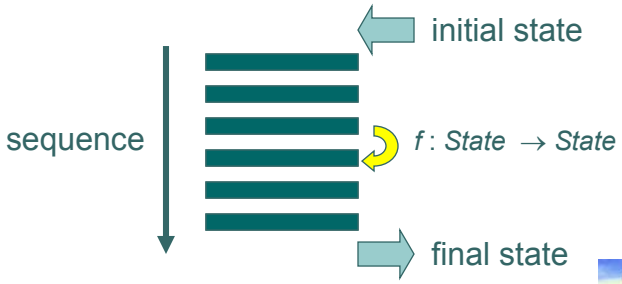
CHESS: Center for Hybrid and Embedded Software Systems

HP Workshop on Advanced Software Technologies
July 20-22
HP Labs, Palo Alto, CA

Standard Software Abstraction (20-th Century Computation)



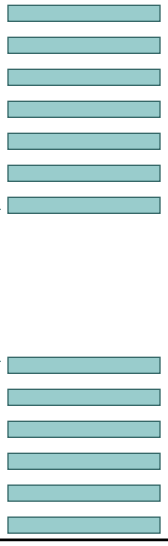
Alan Turing



- Time is irrelevant
- All actions are ordered

Standard Software Abstraction: Processes or Threads

Infinite sequences of state transformations are called "processes" or "threads"



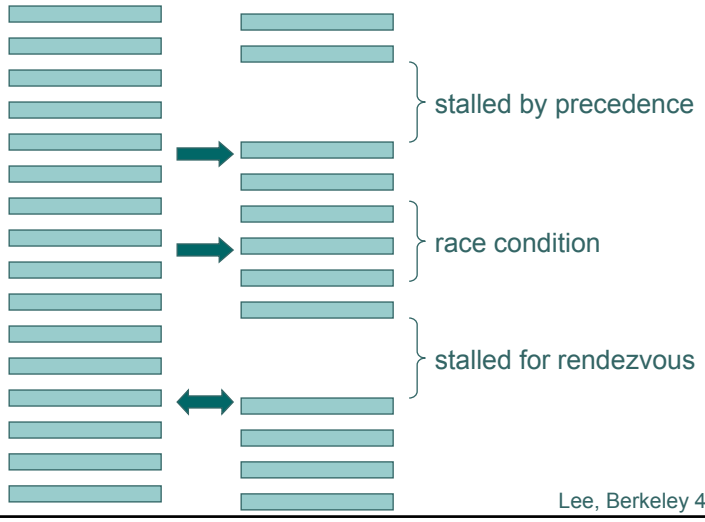
The operating system (typically) provides:

- suspend/resume
- mutual exclusion
- semaphores

Standard Software Abstraction: Concurrency via Interacting Threads

Potential for race conditions, deadlock, and livelock severely compromises software reliability.

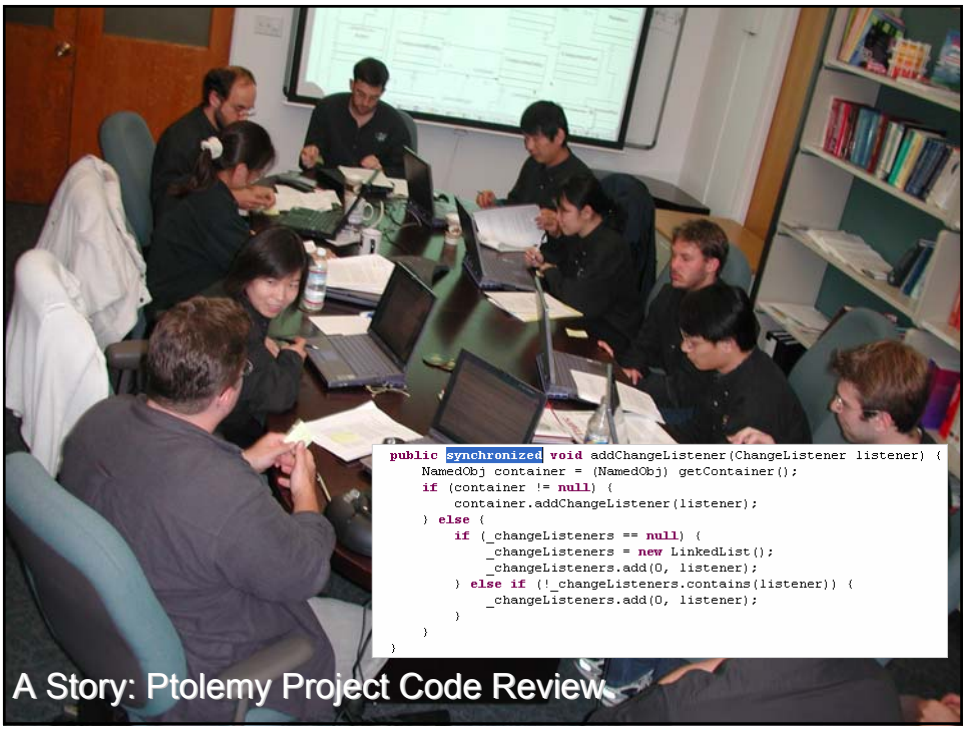
These methods date back to the 1960's (Dijkstra).



A Stake in the Ground

Nontrivial concurrent programs based on processes, threads, semaphores, and mutexes are incomprehensible to humans.

- No amount of process improvement is going to change this.
  - the human brain doesn't work this way.
- Formal methods may help
  - scalability?
  - understandability?
- Better concurrency abstractions will help more



Ptolemy Project Code Review
A Typical Story

- Code review discovers that a method needs to be synchronized to ensure that multiple threads do not reverse each other's actions.
- No problems had been detected in 4 years of using the code.
- Three days after making the change, users started reporting deadlocks caused by the new mutex.
- Analysis and correction of the deadlock is hard.

- But code review successfully identified the flaw.

```
public synchronized void addChangeListener(ChangeListener listener) {
    NamedObj container = (NamedObj) getContainer();
    if (container != null) {
        container.addChangeListener(listener);
    } else {
        if (_changeListeners == null) {
            _changeListeners = new LinkedList();
            _changeListeners.add(0, listener);
        } else if (!_changeListeners.contains(listener)) {
            _changeListeners.add(0, listener);
        }
    }
}
```


Code Review Doesn't Always Work
Another Typical Story

```
/**
CrossRefList is a list that maintains pointers to other CrossRefLists.
…
@author Geroncio Galicia, Contributor: Edward A. Lee
@version $Id: CrossRefList.java,v 1.78 2004/04/29 14:50:00 eal Exp $
@since Ptolemy II 0.2
@Pt.ProposedRating Green (eal)
@Pt.AcceptedRating Green (bart)
*/
public final class CrossRefList implements Serializable  {
    …
    protected class CrossRef implements Serializable{
        …
        // NOTE: It is essential that this method not be
        // synchronized, since it is called by _farContainer(),
        // which is.  Having it synchronized can lead to
        // deadlock.  Fortunately, it is an atomic action,
        // so it need not be synchronized.
        private Object _nearContainer() {
            return _container;
        }

        private synchronized Object _farContainer() {
            if (_far != null) return _far._nearContainer();
            else return null;
        }
        …
    }
}
```

Code that had been in use for four years, central to Ptolemy II, with an extensive test suite, design reviewed to yellow, then code reviewed to green in 2000, causes a deadlock during a demo on April 26, 2004.

And Doubts Remain

```
/**
CrossRefList is a list that maintains pointers to other CrossRefLists.
…
@author Geroncio Galicia, Contributor: Edward A. Lee
@version $Id: CrossRefList.java,v 1.78 2004/04/29 14:50:00 eal Exp $
@since Ptolemy II 0.2
@Pt.ProposedRating Green (eal)
@Pt.AcceptedRating Green (bart)
*/
public final class CrossRefList implements Serializable  {
    …
    protected class CrossRef implements Serializable{
        …
        private synchronized void _dissociate() {
            _unlink(); // Remove this.
            // NOTE: Deadlock risk here!  If _far is waiting
            // on a lock to this CrossRef, then we will get
            // deadlock. However, this will only happen if
            // we have two threads simultaneously modifying a
            // model. At the moment (4/29/04), we have no
            // mechanism for doing that without first
            // acquiring write permission the workspace().
            // Two threads cannot simultaneously hold that
            // write access.
            if (_far != null) _far._unlink(); // Remove far
        }
    }
```

Safety of this code depends on policies maintained by entirely unconnected classes. The language and synchronization mechanisms provide no way to talk about these systemwide properties.

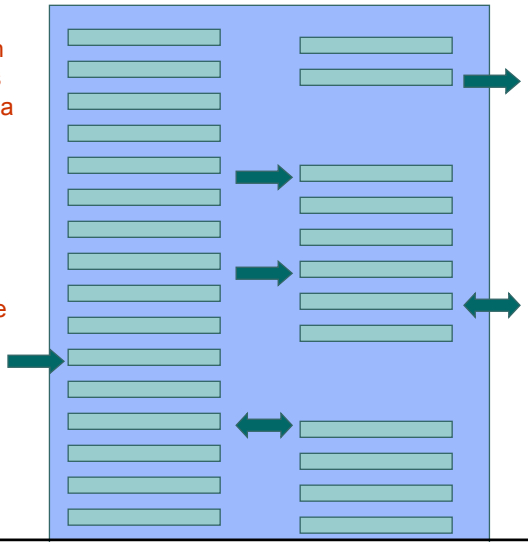
What it Feels Like to Use the *synchronized* Keyword in Java

Image "borrowed" from an Iomega advertisement for Y2K software and disk drives, *Scientific American*, September 1999.

Diagnosis: Interacting Processes are Not Compositional

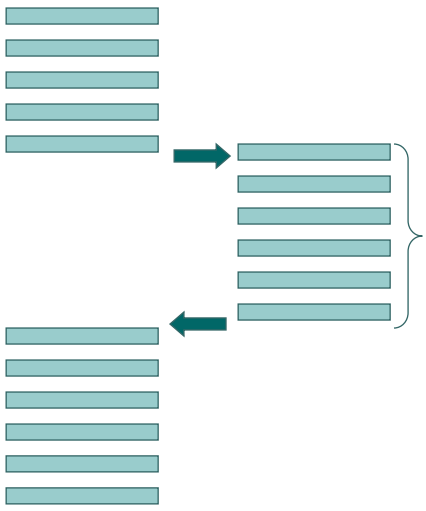
An aggregation of processes is not a process (a total order of external interactions). What is it?

Many software failures are due to this ill-defined composition.



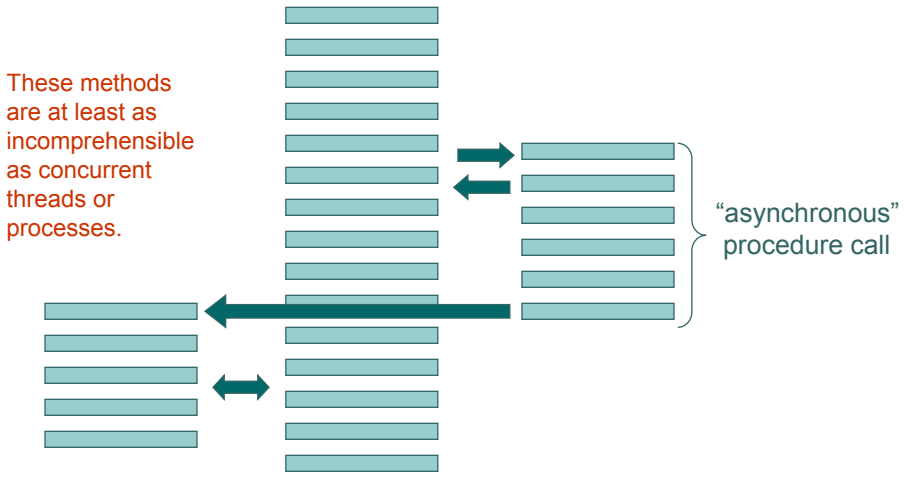
Distributed Version of 20-th Century Computation

Force-fitting the sequential abstraction onto parallel hardware.

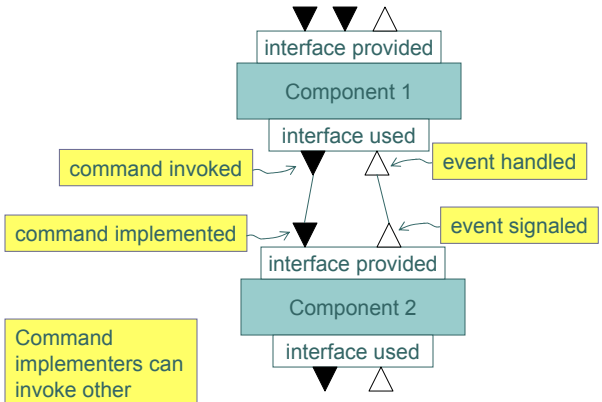


Combining Processes and RPC –

Split-Phase Execution, Futures,
Asynchronous Method Calls, Callbacks, …



Model Used in Wireless Sensor Nets
No Threads: nesC and TinyOS

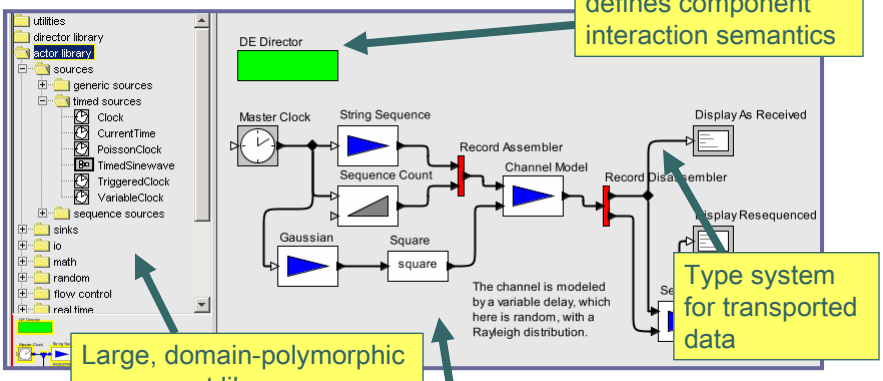


Typical usage pattern:

- hardware interrupt signals an event.
- event handler posts a task.
- tasks are executed when machine is idle.
- tasks execute atomically w.r.t. one another.
- tasks can invoke commands and signal events.
- hardware interrupts can interrupt tasks.
- exactly one monitor, implemented by disabling interrupts.

Ptolemy II: Framework for Experimenting with Alternative Concurrent Models of Computation

Basic Ptolemy II infrastructure:



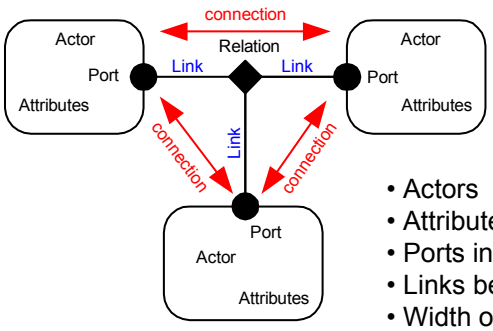
Director from a library defines component interaction semantics

Type system for transported data

Large, domain-polymorphic component library.

Visual editor supporting an abstract syntax

The Basic Abstract Syntax

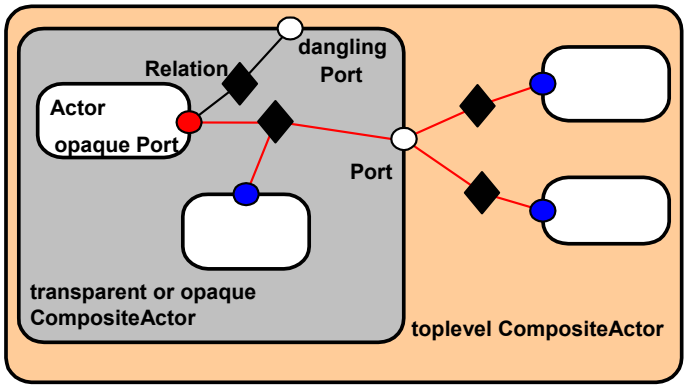


- Actors
- Attributes on actors (parameters)
- Ports in actors
- Links between ports
- Width on links (channels)
- Hierarchy

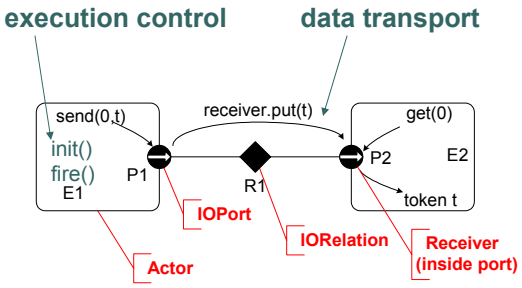
Concrete syntaxes:

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

Hierarchy - Composite Components



Abstract Semantics
of *Actor-Oriented* Models of Computation

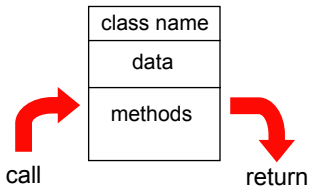


Actor-Oriented Models of Computation that we have implemented:

- 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)
- …

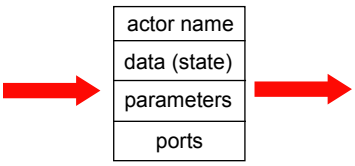
What is an *Actor-Oriented* MoC?

Traditional component interactions:



What flows through an object is sequential control

Actor oriented:



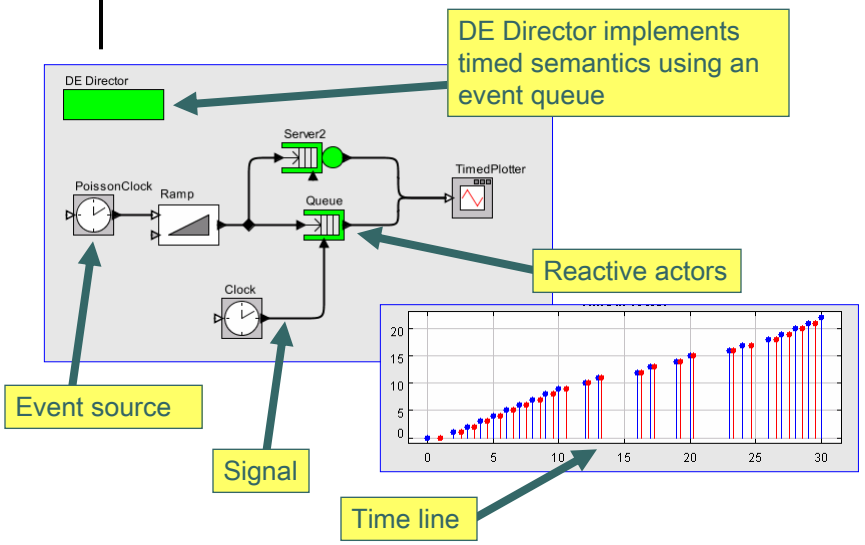
What flows through an object is streams of data

Models of Computation Implemented in Ptolemy II

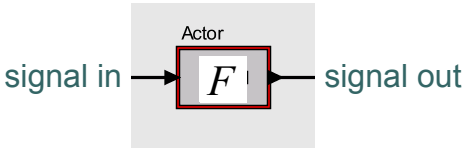
- CI – Push/pull component interaction
- Click – Push/pull with method invocation
- CSP – concurrent threads with rendezvous
- CT – continuous-time modeling
- DE – discrete-event systems
- DDE – distributed discrete events
- FSM – finite state machines
- DT – discrete time (cycle driven)
- Giotto – synchronous periodic
- GR – 2-D and 3-D graphics
- PN – process networks
- DPN – distributed process networks
- SDF – synchronous dataflow
- SR – synchronous/reactive
- TM – timed multitasking

Most of these are actor oriented.

Discrete Event Models



Semantics of DE Signals



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

A signal is a partial function:

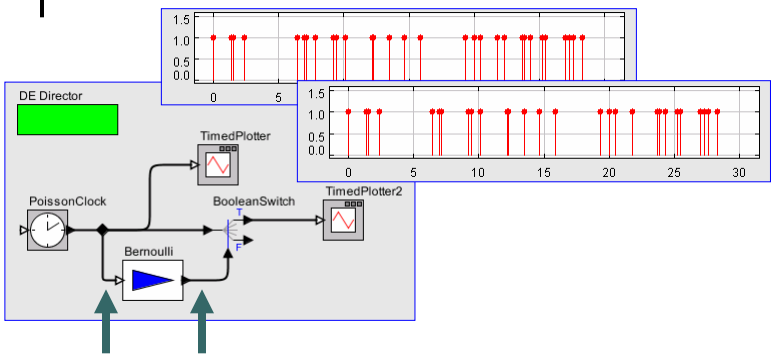
$$F : \Re \times I \rightarrow T$$

Data type (set of values)

Real numbers (approximated by doubles)

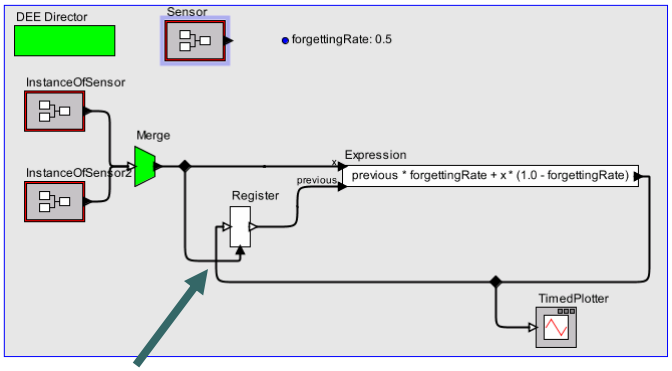
Integers (allowing for simultaneous events in a signal)

Subtleties: Simultaneous Events



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

Subtleties: Feedback



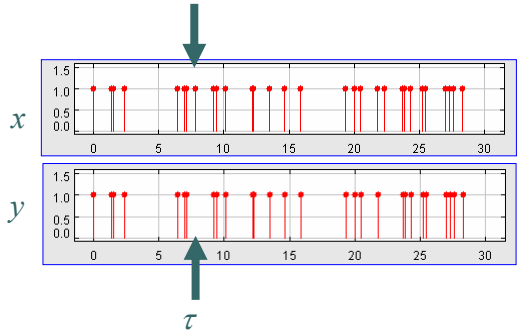
Data precedence analysis has to take into account the non-strictness of this actor (that an output can be produced despite the lack of an input).

Discrete-Event Semantics

Cantor metric:

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

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



Causality

Causal:

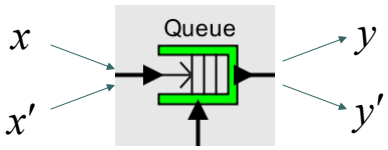
$$d(y, y') \leq d(x, x')$$

Strictly causal:

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

Delta causal:

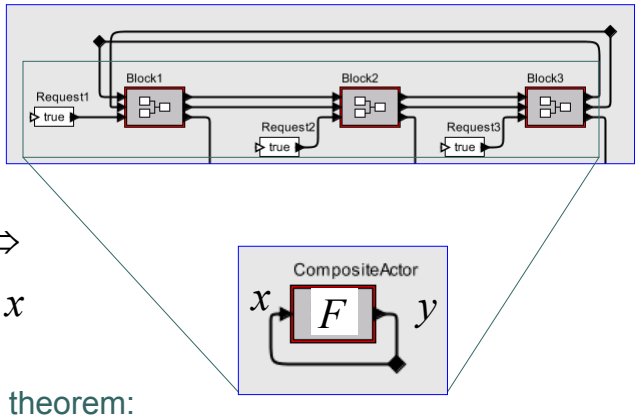
$$\exists \delta < 1,$$
$$d(y, y') \leq \delta\, d(x, x')$$



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

Semantics of Composition

If the components are deterministic, the composition is deterministic.

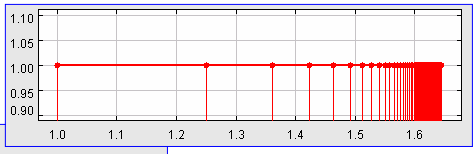


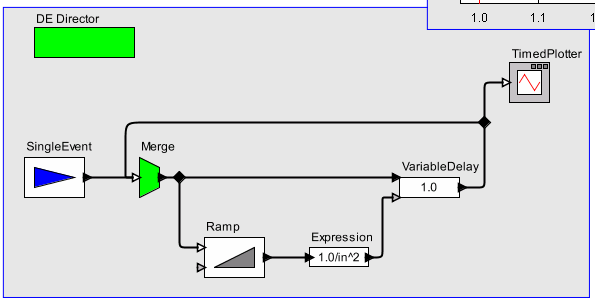
$$x = y \Rightarrow$$
$$F(x) = x$$

Banach fixed point theorem:

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

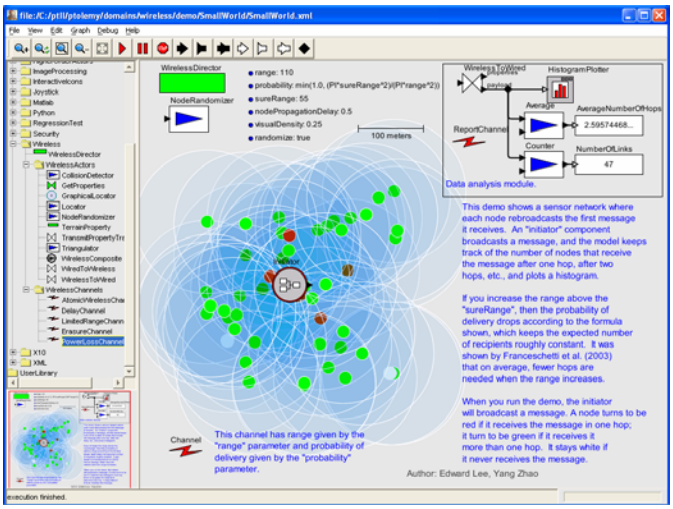
Zeno Systems





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

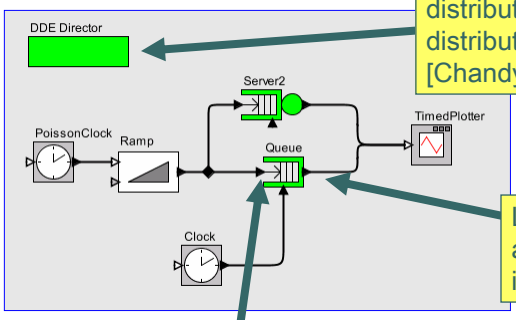
Extension of Discrete-Event Modeling for Wireless Sensor Nets



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.

Distributed Discrete Event Models as Currently Implemented in Ptolemy II



DDE Director supports distributed execution and a distributed notion of time [Chandy & Misra 1979]

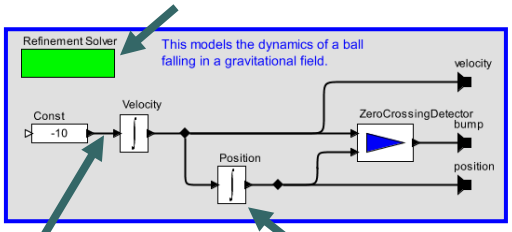
Local notion of time in each actor, advancing only when input is received

Blocking read at input ports prevents time from locally advancing without "permission" from a source

This is the "Chandy and Misra" style of distributed discrete events [1979], which compared to Croquet and Time Warp [Jefferson, 1985], is "conservative."

Continuous-Time Models

Director implements a “solver” that constructs an approximation to the continuous-time behavior.

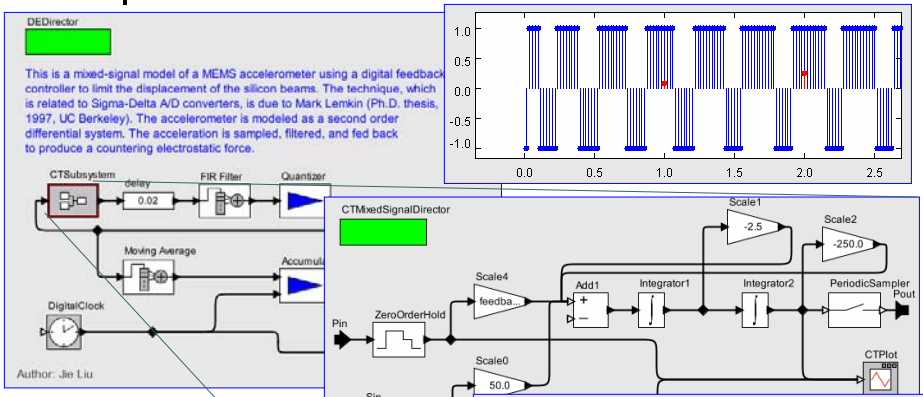


A signal has a value at all real-valued times.

Integrator used to define systems governed by ordinary differential equations.

Heterogeneous Models
Mixed Signals: DE + CT

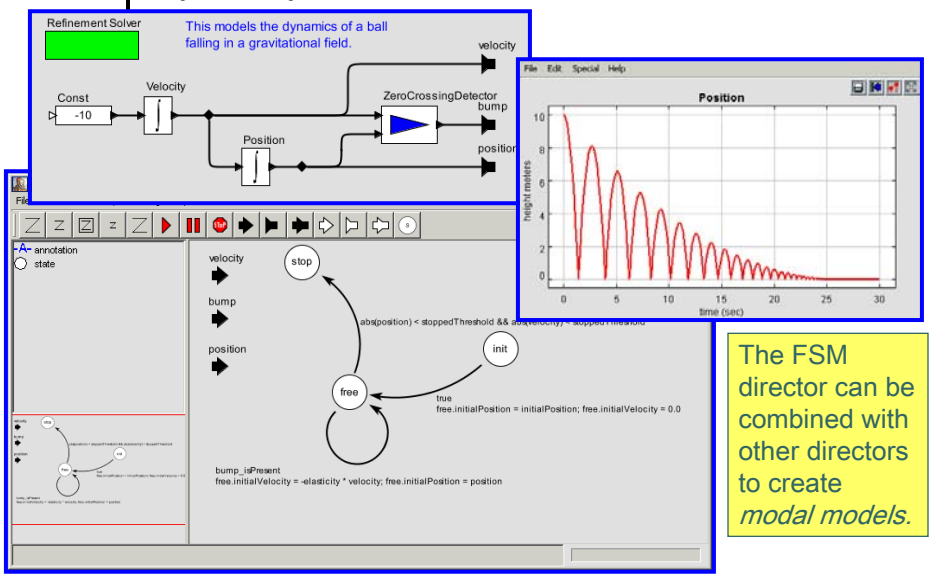
DE signal



DE model of a digital controller

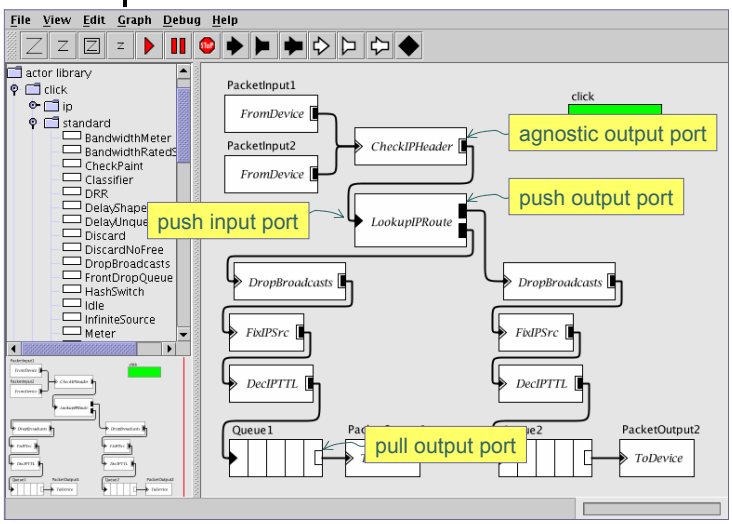
CT model of mechanical system

Heterogeneous Models
Hybrid Systems: CT + FSM



The FSM director can be combined with other directors to create *modal models.*

Untimed Concurrency Model:
First Example: Click

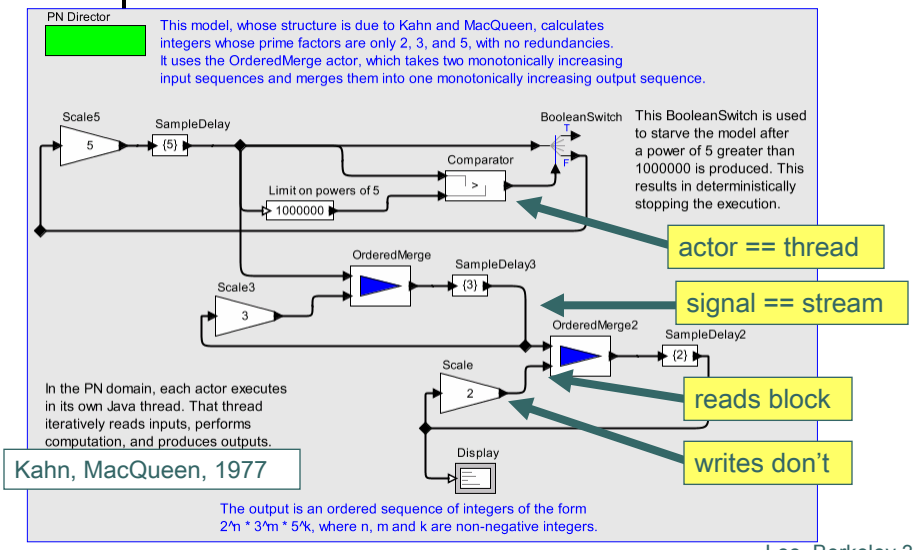


Implementation of Click with a visual syntax in Mescal (Keutzer, et al.)

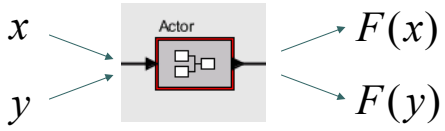
Typical usage pattern:

- queues have push input, pull output.
- schedulers have pull input, push output.
- thin wrappers for hardware have push output or pull input only.
- push or pull handled by method calls

Untimed Concurrency Model: Second Example: Process Networks



PN Semantics



- A signal is a sequence of values
- Define a prefix order:

$$x \sqsubset y$$

means that x is a prefix of y.

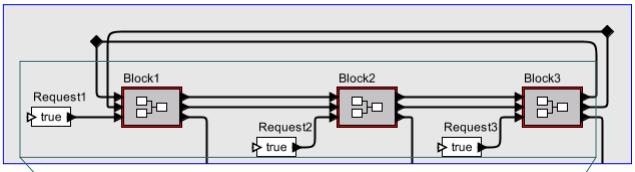
- Actors are *monotonic* functions:

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

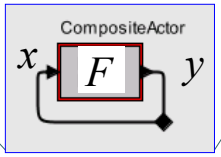
- Stronger condition: Actors are *continuous* functions (intuitively: they don't wait forever to produce outputs).

PN Semantics of Composition (Kahn, '74)

If the components are deterministic, the composition is deterministic.



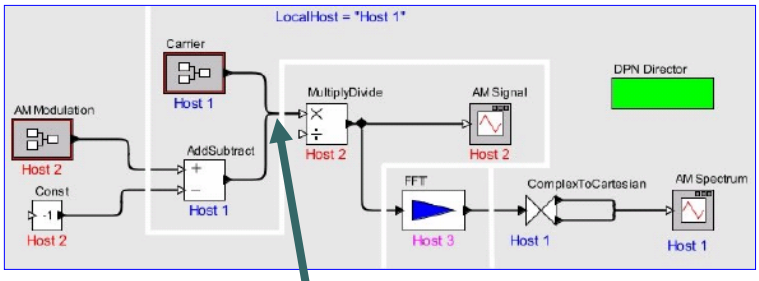
$$x = y \Rightarrow$$
$$F(x) = x$$



Knaster-Tarski fixed point theorem:

- Continuous function has a unique least fixed point
- Execution procedure for finding that fixed point
- Successive approximations to the fixed point

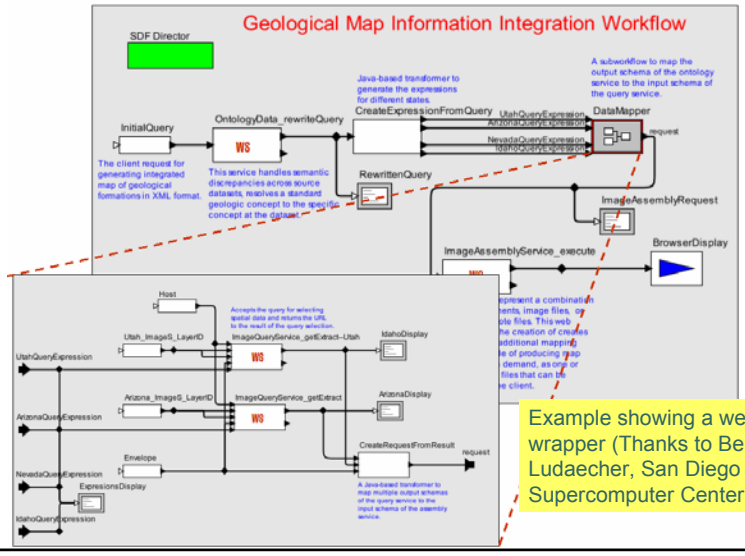
Distributed Process Networks



Transport mechanism between hosts is provided by the director. Transparently provides guaranteed delivery and ordered messages.

Created by Dominique Ragot, Thales Communications

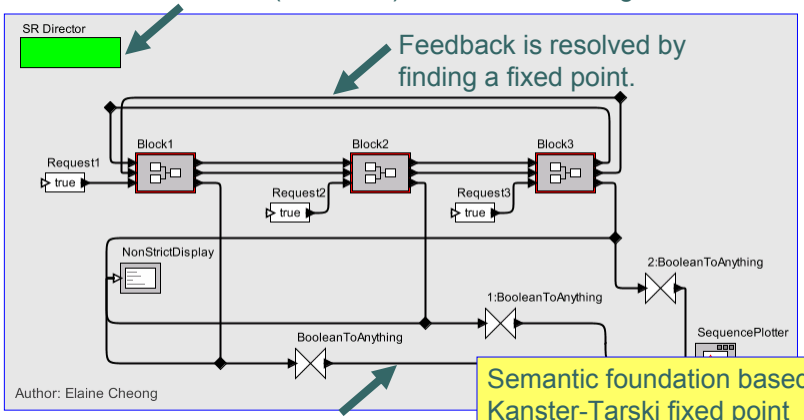
Kepler: Extensions to Ptolemy II for Scientific Workflows



Example showing a web service wrapper (Thanks to Bertram Ludaecher, San Diego Supercomputer Center)

Synchronous Models of Computation

Director finds a value (or *absent*) at each "tick" of a global clock



Signal has a value or is *absent* at each tick of the "clock."

Semantic foundation based on Kanster-Tarski fixed point theorem on Scott topologies.

Languages Based on the Synchronous Model of Computation

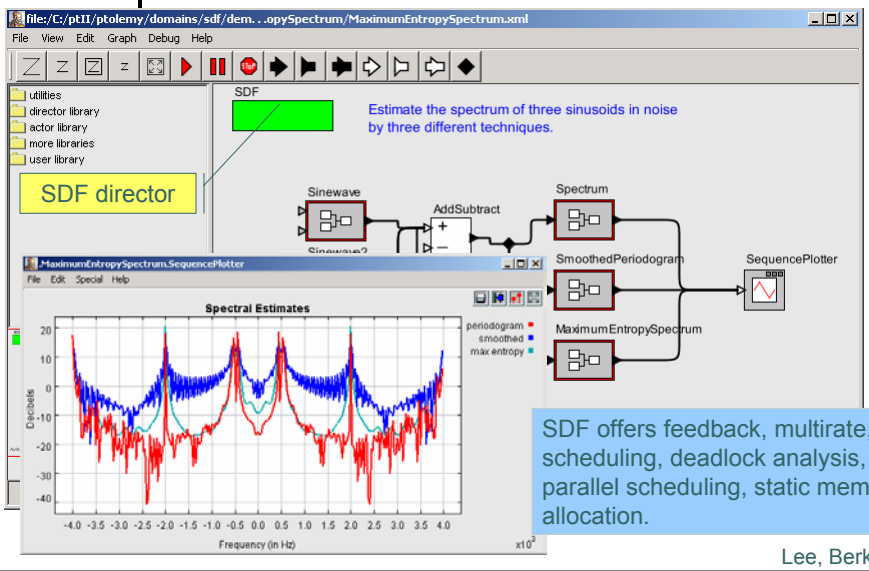
- Lustre (and SCADE)
- Esterel
- Signal
- Statecharts (and UML state machines)
- Argos
- …

Dataflow Models of Computation

- Computation graphs [Karp & Miller - 1966]
- Process networks [Kahn - 1974]
- Static dataflow [Dennis - 1974]
- Dynamic dataflow [Arvind, 1981]
- K-bounded loops [Culler, 1986]
- Synchronous dataflow [Lee & Messerschmitt, 1986]
- Structured dataflow [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 and Parks, 1995]
- Heterochronous dataflow [Girault, Lee, & Lee, 1997]
- …

Many tools, software frameworks, and hardware architectures have been built to support one or more of these.

Synchronous Dataflow (SDF)
(Lee and Messerschmitt, 1986)



SDF offers feedback, multirate, static scheduling, deadlock analysis, parallel scheduling, static memory allocation.

Synchronous Dataflow (SDF)
Fixed Production/Consumption Rates

- Balance equations (one for each channel):

$$f_A N = f_B M$$

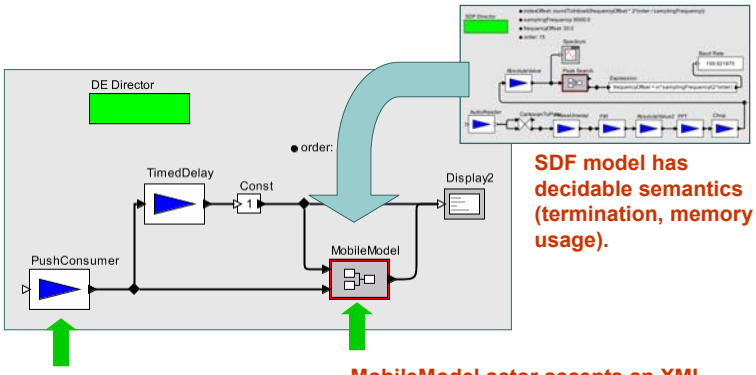
number of tokens consumed

number of firings per "iteration"

number of tokens produced

- Schedulable statically
- Get a well-defined "iteration"
- Decidable:
  - buffer memory requirements
  - deadlock

One Consequence of SDF Semantics: Mobile Code that Cannot Perform Denial of Service Attacks



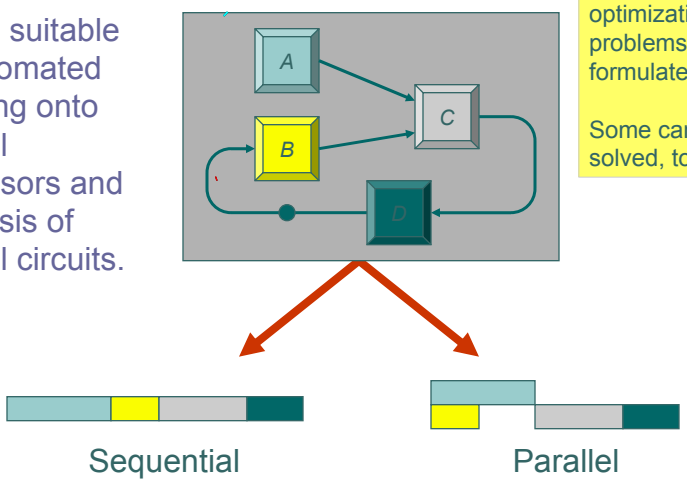
SDF model has decidable semantics (termination, memory usage).

PushConsumer actor receives pushed data provided via CORBA, where the data is an XML model of a signal analysis algorithm.

MobileModel actor accepts an XML description of a model. It then executes that model on a stream of input data using locally defined component implementations.

Parallel Scheduling of SDF Models

SDF is suitable for automated mapping onto parallel processors and synthesis of parallel circuits.

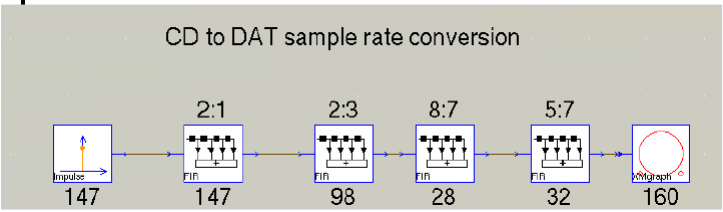


Many scheduling optimization problems can be formulated.

Some can be solved, too!

Scheduling Tradeoffs

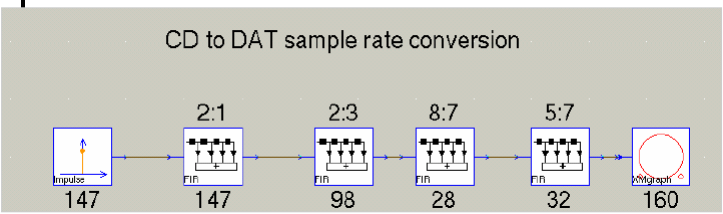
(Bhattacharyya, Parks, Pino)



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

Source: Shuvra Bhattacharyya

Minimum Buffer Schedule



ABABCABCABABCABCDEAFFFFFBABCABCABABCDE
AFFFFFBCABABCABCABABCDEAFFFFFBCABABCABC
DEAFFFFFBABCABCABABCABCDEAFFFFFBABCABCA
BABCDEAFFFFFBCABABCABCABABCDEAFFFFFEBCA
FFFFFBABCABCDEAFFFFFBABCABCABABCABCDEAF
FFFFBABCABCABABCDEAFFFFFBCABABCABCABABC
DEAFFFFFBCABABCABCDEAFFFFFBABCABCABABCA
BCDEAFFFFFBABCABCABABCDEAFFFFFEBCAFFFFFB
ABCABCABABCDEAFFFFFBCABABCABCDEAFFFFFBA
BCABCABABCABCDEAFFFFFBABCABCABABCDEAFFF
FFBCABABCABCABABCDEAFFFFFBCABABCABCDEAF
FFFFBABCABCABABCABCDEAFFFFFEBAFFFFFBCABC
ABABCDEAFFFFFBCABABCABCABABCDEAFFFFFBCA
BABCABCDEAFFFFFBABCABCABABCABCDEAFFFFFB
ABCABCABABCDEAFFFFFBCABABCABCABABCDEAF
FFFFBCABABCABCDEFFFFFEFFFFF

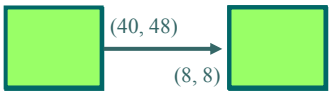
Source: Shuvra Bhattacharyya

Selected Generalizations

- Multidimensional Synchronous Dataflow (1993)
  - Arcs carry multidimensional streams
  - One balance equation per dimension per arc
- Cyclo-Static Dataflow (Lauwereins, et al., 1994)
  - Periodically varying production/consumption rates
- Boolean & Integer Dataflow (1993/4)
  - Balance equations are solved symbolically
  - Permits data-dependent routing of tokens
  - Heuristic-based scheduling (undecidable)
- Dynamic Dataflow (1981-)
  - Firings scheduled at run time
  - Challenge: maintain bounded memory, deadlock freedom, liveness
  - Demand driven, data driven, and fair policies all fail
- Kahn Process Networks (1974-)
  - Replace discrete firings with process suspension
  - Challenge: maintain bounded memory, deadlock freedom, liveness
- Heterochronous Dataflow (1997)
  - Combines state machines with SDF graphs
  - Very expressive, yet decidable

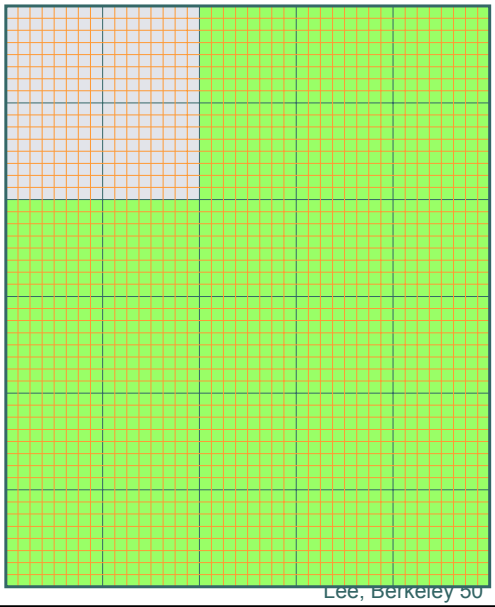
Multidimensional SDF
(Lee, 1993)

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

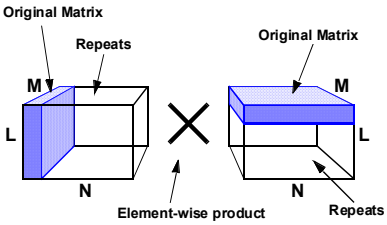


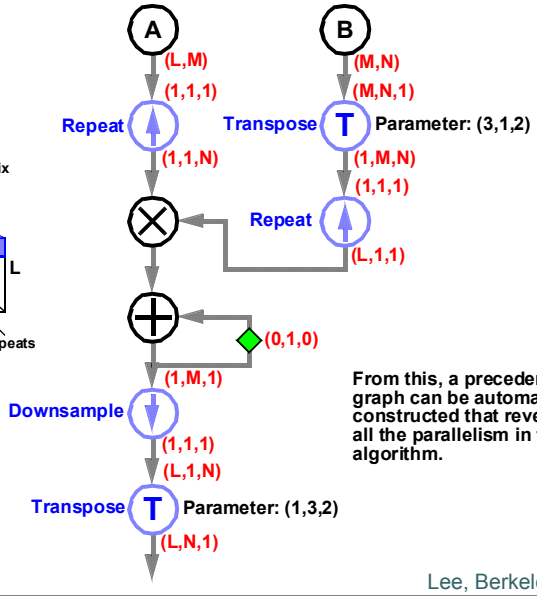
- Balance equations and scheduling policies generalize.
- Much more data parallelism is exposed.

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



MDSDF Structure Exposes Fine-Grain Data Parallelism





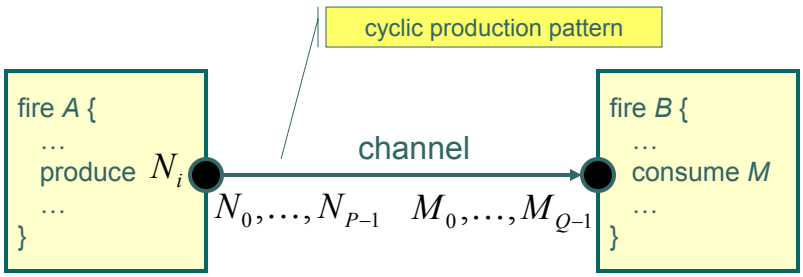
From this, a precedence graph can be automatically constructed that reveals all the parallelism in the algorithm.

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

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

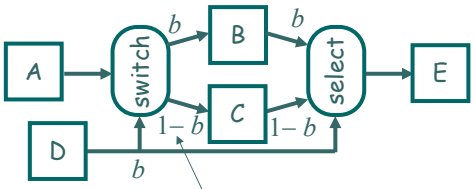
- Actors cycle through a regular production/consumption pattern.
- Balance equations become:

$$f_A \sum_{i=0}^{R-1} N_{i \bmod P} = f_B \sum_{i=0}^{R-1} M_{i \bmod Q};\ R = lcm(P,Q)$$



Boolean and Integer Dataflow (BDF, IDF)
(Lee and Buck, 1993)

- Balance equations are solved symbolically in terms of unknowns that become known at run time.
- An *annotated schedule* is constructed with predicates guarding each action.
- Existence of such an annotated schedule is undecidable (as is deadlock & bounded memory)



$$f_{switch} b = f_B$$
$$f_{switch}(1-b) = f_C$$
$$\ldots$$

Production rate is unknown and is represented symbolically by a variable (*b*).

Dynamic Dataflow (DDF)

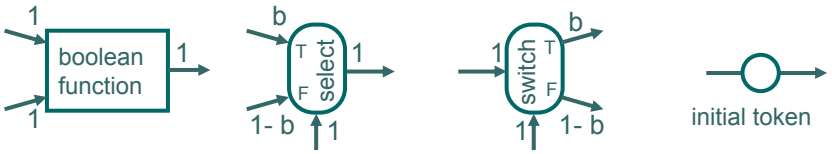
- Actors have *firing rules*
  - Set of finite prefixes on input sequences
  - For determinism: No two such prefixes are joinable under a prefix order
  - Firing function applied to finite prefixes yield finite outputs
- Scheduling objectives:
  - Do not stop if there are executable actors
  - Execute in bounded memory if this is possible
  - Maintain determinacy if possible
- Policies that fail:
  - Data-driven execution
  - Demand-driven execution
  - Fair execution
  - Many balanced data/demand-driven strategies
- Policy that succeeds (Parks 1995):
  - Execute with bounded buffers
  - Increase bounds only when deadlock occurs

DDF, like BDF and IDF is undecidable (deadlock, bounded memory, schedule)

Undecidability
(Buck '93)

- Sufficient set of actors for undecidability:
  - boolean functions on boolean tokens
  - switch and select
  - initial tokens on arcs

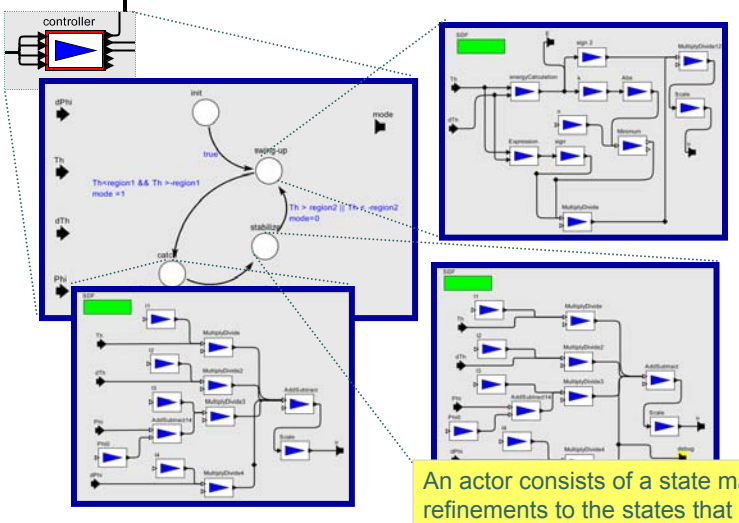
These four parts are sufficient to build any computable function.



- Undecidable:
  - deadlock
  - bounded buffer memory
  - existence of an annotated schedule

BDF, IDF, DDF, and PN are all undecidable in this sense. Fortunately, we can identify a large decidable subset, which we call heterochronous dataflow (HDF).

Example of a Heterochronous Dataflow Model



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

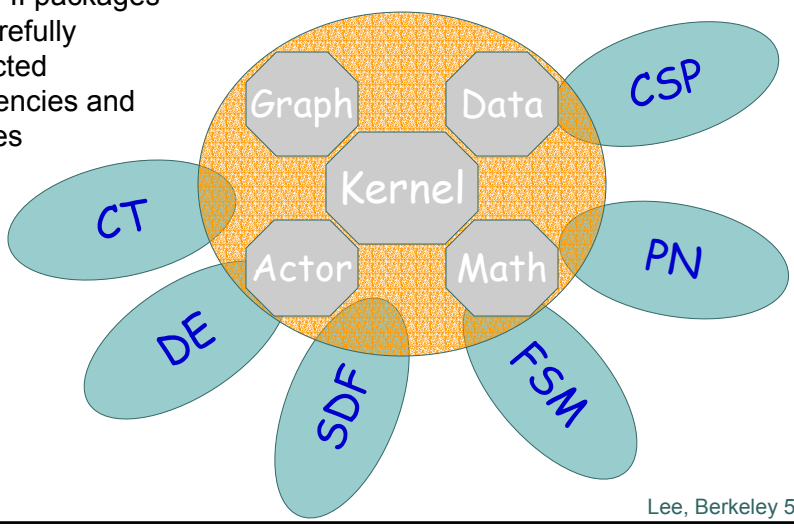
Heterochronous Dataflow (HDF)
(Girault, Lee, and Lee, 1997)

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

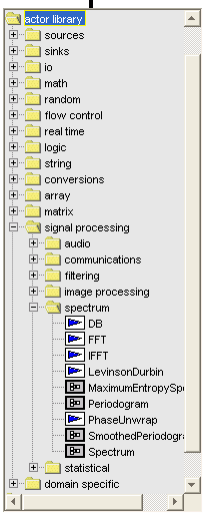
Related to "parameterized dataflow" of Bhattachrya and Bhattacharyya (2001).

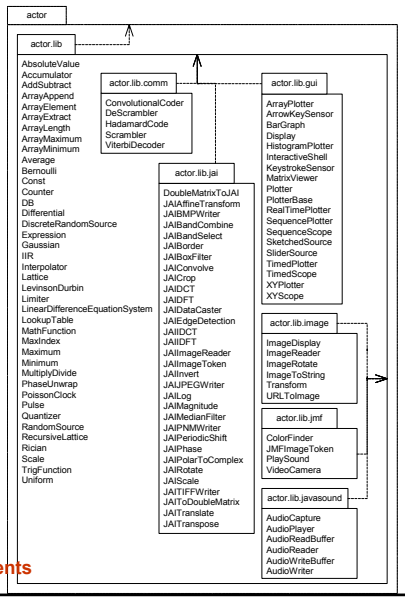
Ptolemy II Software Architecture Built for Extensibility

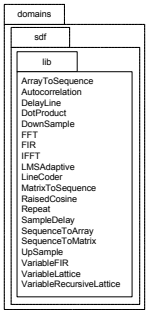
Ptolemy II packages have carefully constructed dependencies and interfaces



Ptolemy II Component Library



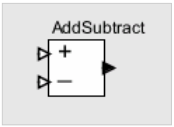




UML package diagram of key actor libraries included with Ptolemy II.

- **Data polymorphic components**
- **Behaviorally polymorphic components**

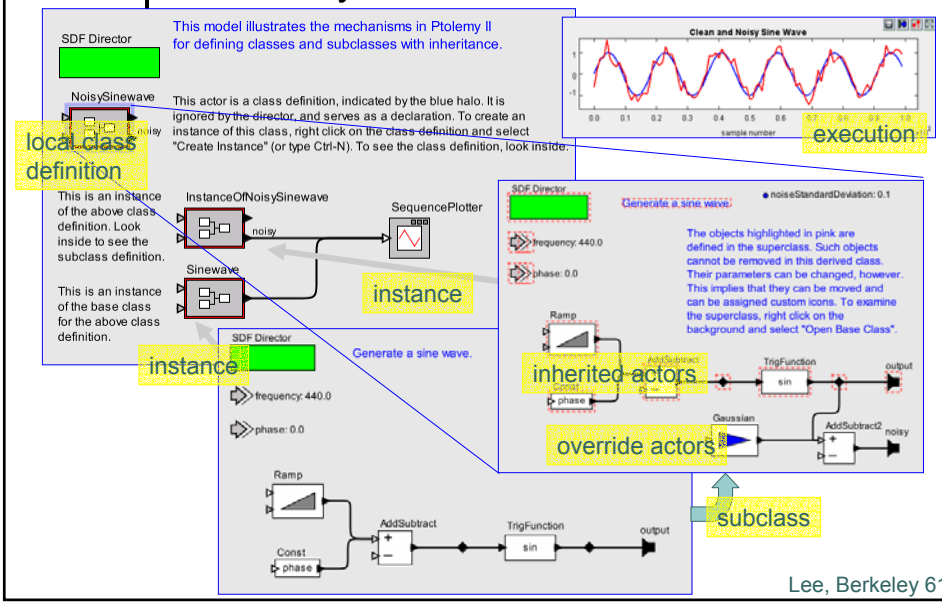
Polymorphic Components - Component Library Works Across Data Types and Domains



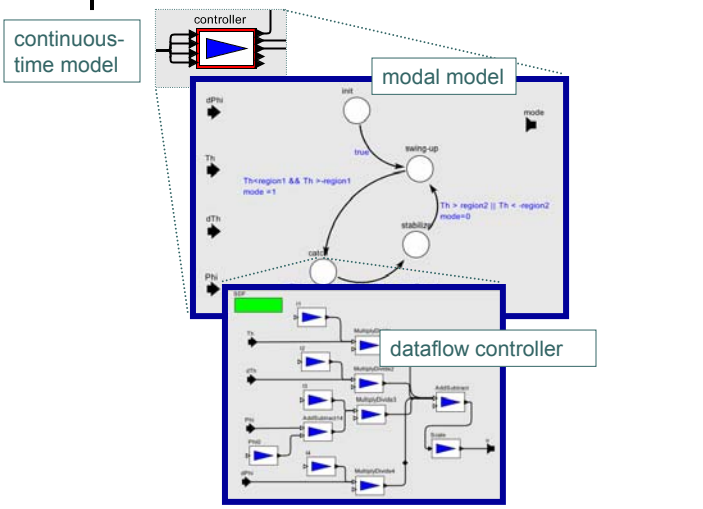
- Data polymorphism:
  - Add numbers (int, float, double, Complex)
  - Add strings (concatenation)
  - Add composite types (arrays, records, matrices)
  - Add user-defined types
- Behavioral polymorphism:
  - In dataflow, add when all connected inputs have data
  - In a time-triggered model, add when the clock ticks
  - In discrete-event, add when any connected input has data, and add in zero time
  - In process networks, execute an infinite loop in a thread that blocks when reading empty inputs
  - In CSP, execute an infinite loop that performs rendezvous on input or output
  - In push/pull, ports are push or pull (declared or inferred) and behave accordingly
  - In real-time CORBA, priorities are associated with ports and a dispatcher determines when to add

By not choosing among these when defining the component, we get a huge increment in component re-usability. But how do we ensure that the component will work in all these circumstances?

Shared Infrastructure Modularity Mechanisms



More Shared Infrastructure: Hierarchical Heterogeneity and Modal Models



example Ptolemy II model: hybrid control system

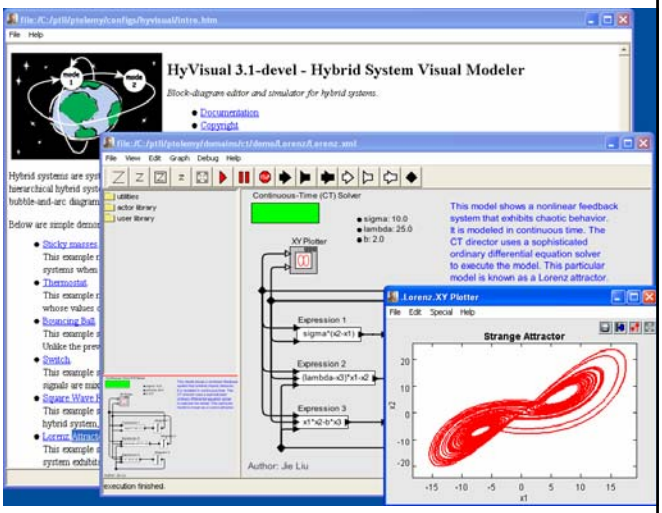
Branding

Ptolemy II *configurations* are Ptolemy II models that specify

- welcome window
- help menu contents
- library contents
- File->New menu contents
- default model structure
- etc.

A configuration can identify its own "brand" independent of the "Ptolemy II" name and can have more targeted objectives.

An example is HyVisual, a tool for hybrid system modeling. VisualSense is another tool for wireless sensor network modeling.

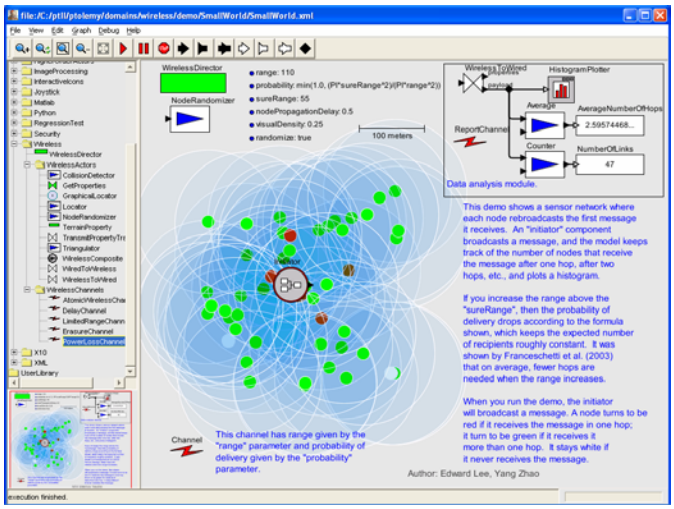


Ptolemy II Extension Points

- 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.

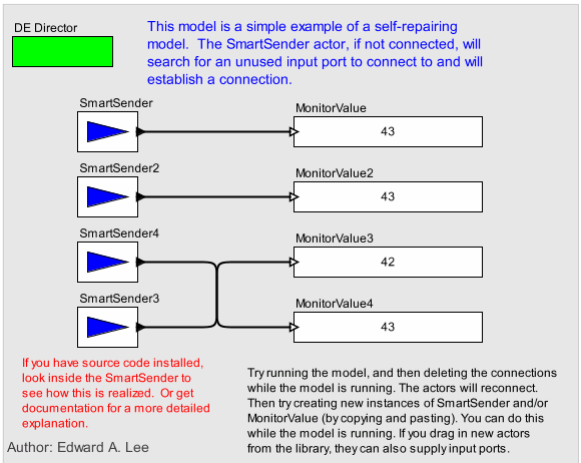
Example Extension: VisualSense



- Branded
- Customized visualization
- Customized model of computation (an extension of DE)
- Customized actor library
- Motivated some extensions to the core (e.g. classes, icon editor).

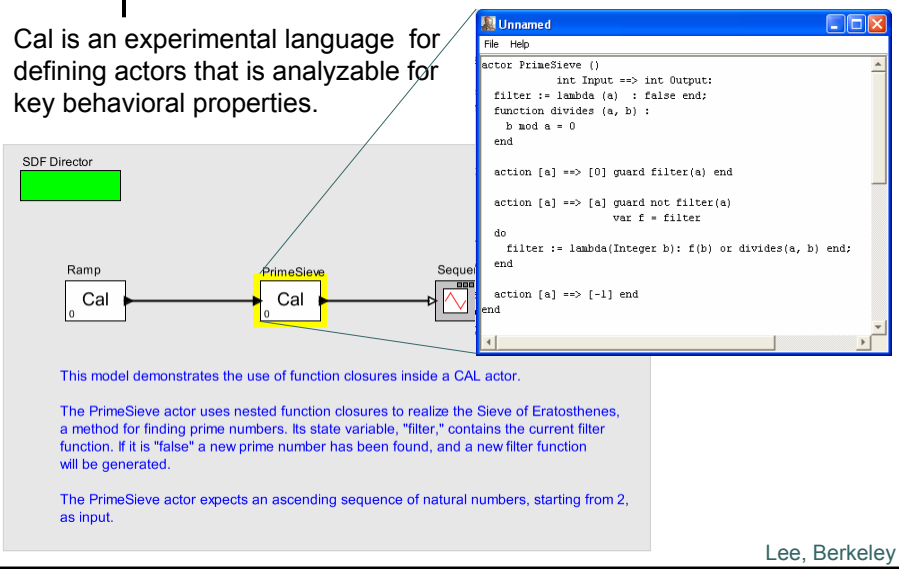
Example Extensions: Self-Repairing Models

Concept demonstration built together with Boeing to show how to write actors that adaptively reconstruct connections when the model structure changes.

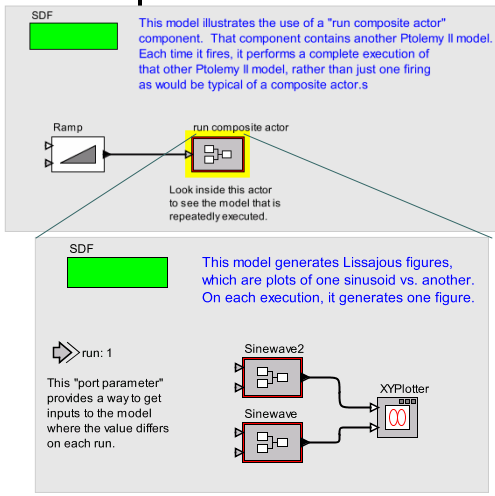


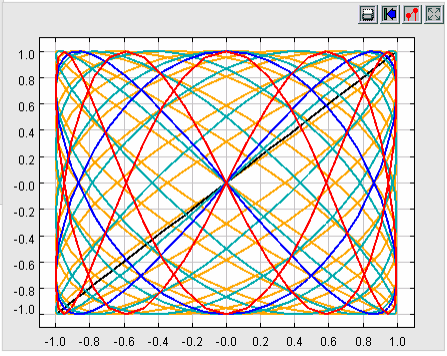
Example Extensions
Python Actors and Cal Actors

Cal is an experimental language for defining actors that is analyzable for key behavioral properties.



Example Extensions
Using Models to Control Models

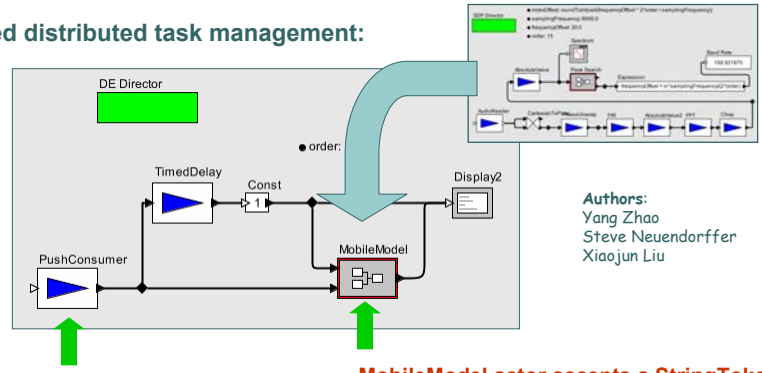




This is an example of a “higher-order component,” or an actor that references one or more other actors.

Examples of Extensions
Mobile Models

Model-based distributed task management:

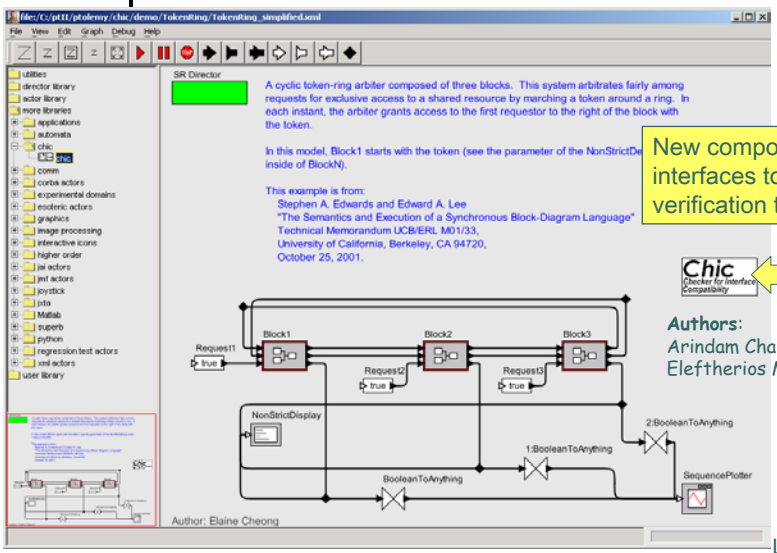


Authors:
Yang Zhao
Steve Neuendorffer
Xiaojun Liu

PushConsumer actor receives pushed data provided via CORBA, where the data is an XML model of a signal analysis algorithm.

MobileModel actor accepts a StringToken containing an XML description of a model. It then executes that model on a stream of input data.

Examples of Extensions
Hooks to Verification Tools

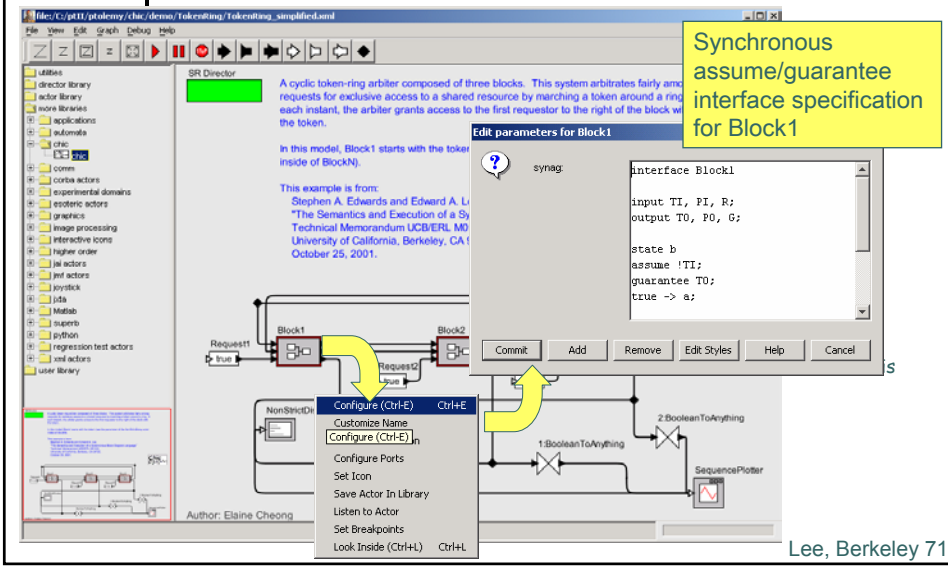


New component interfaces to Chic verification tool

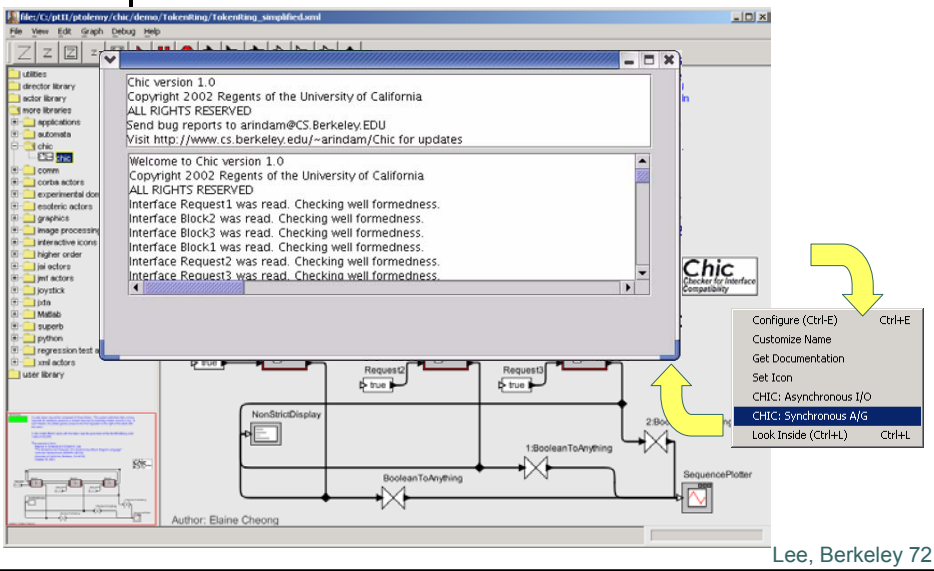
Chic
Checker for Interface Compatibility

Authors:
Arindam Chakrabarti
Eleftherios Matsikoudis

Examples of Extensions
Hooks to Verification Tools



Examples of Extensions
Hooks to Verification Tools



Conclusion

- Threads suck
- There are many alternative concurrent MoCs
- The ones you know are the tip of the iceberg
- Ptolemy II is a lab for experimenting with them
- Specializing MoCs can be useful
- Mixing specialized MoCs can be useful.