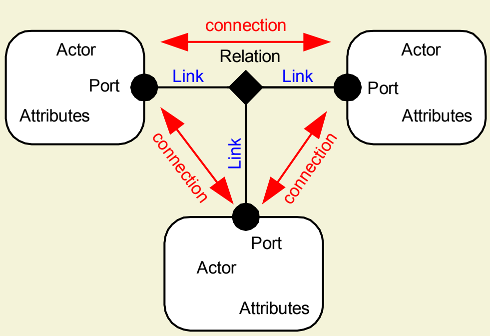
Embedded Software from Concurrent Component Models

Edward A. Lee
UC Berkeley

with
**Shuvra Bhattacharyya, Johan Eker,
Christopher Hylands, Jie Liu, Xiaojun Liu,
Steve Neuendorffer, Jeff Tsay, and Yuhong Xiong**

ACM SIGPLAN 2001 Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'2001) Jun 22-23, 2001, Snowbird, Utah, USA

View of SW Architecture: *Actors* with *Ports* and *Attributes*



Model of Computation:

- Messaging schema
- Flow of control
- Concurrency

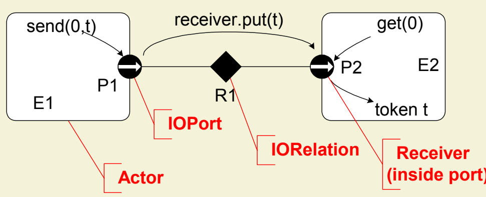
Examples:

- Time triggered
- Process networks
- Discrete-event systems
- Dataflow systems
- Publish & subscribe

Key idea: The model of computation is part of the framework within which components are embedded not part of the components themselves.

Actor View of Producer/Consumer Components

Basic Transport:



Models of Computation:

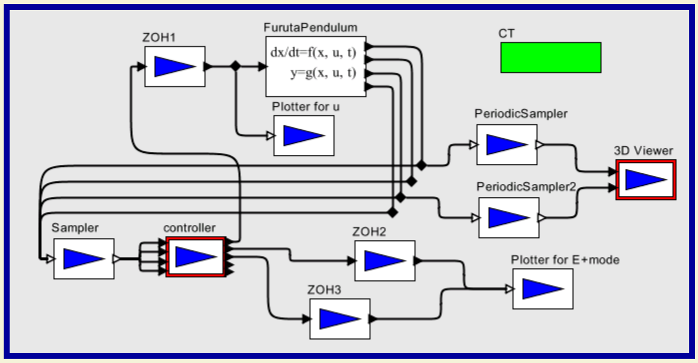
- continuous-time
- dataflow
- rendezvous
- discrete events
- synchronous
- time-driven
- publish/subscribe
- …

Examples of Actors+Ports Software Architectures

- Simulink (The MathWorks)
- Labview (National Instruments)
- OCP, open control platform (Boeing)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- VHDL, Verilog, SystemC (Various)
- Polis & Metropolis (UC Berkeley)
- Ptolemy & Ptolemy II (UC Berkeley)
- …

Model by Johan Eker

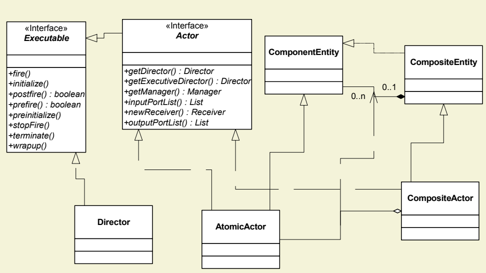
What a Program Looks Like



Ptolemy II model of an embedded control system and the system being controlled. This is a hierarchical, heterogeneous model that combines four models of computation.

Contrast with Object Orientation

- Call/return imperative semantics
  - band-aids: futures, proxies, monitors
- Poorly models the environment
  - which does not have call/return semantics
- Concurrency is via ad-hoc calling conventions
- Nothing at all to say about time

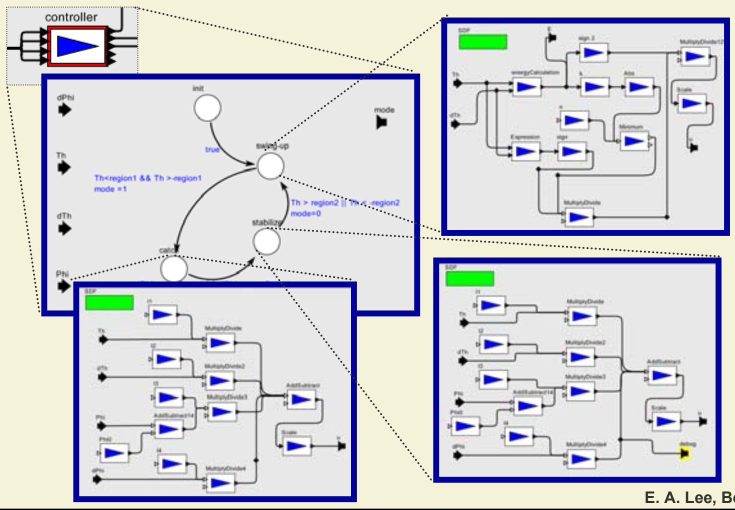


Object modeling emphasizes inheritance and procedural interfaces.

Actor modeling emphasizes concurrency and communication abstractions.

Hierarchy and Heterogeneity 1
Modal Models

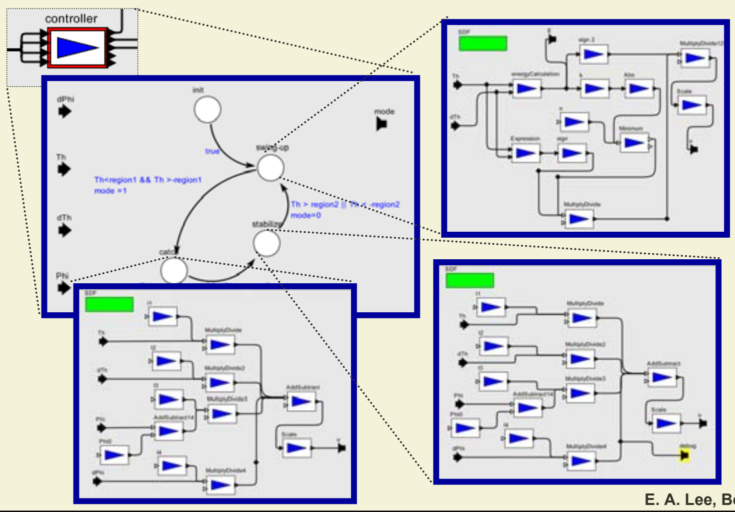
Model by Johan Eker



Domains

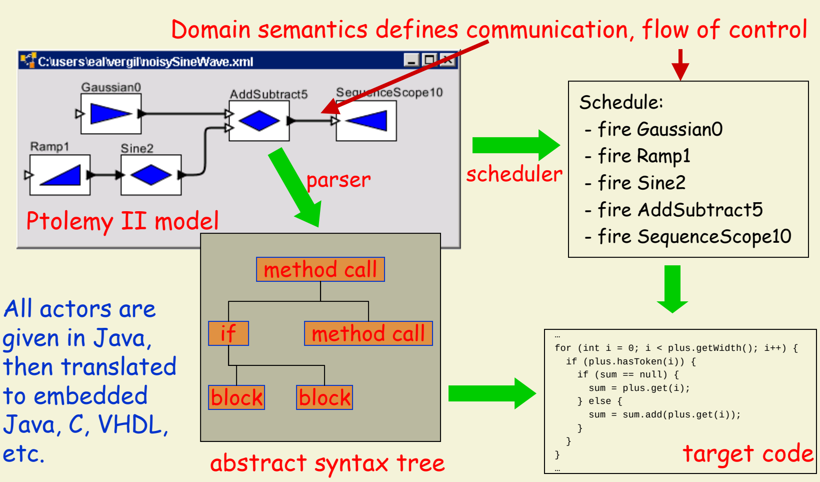
- Each level of the hierarchy may have its own "laws of physics"
  - communication semantics
  - flow of control constraints
- Domain
  - a region of the universe where a certain set of "laws of physics" applies
  - Realizes a "model of computation"

A Problem: Compiling these Models: "Code generation"



Outline of our Approach

Jeff Tsay,
Christopher Hylands,
Steve Neuendorffer



All actors are given in Java, then translated to embedded Java, C, VHDL, etc.

Division of Responsibility

- Domain semantics defines
  - flow of control across actors
  - communication protocols between actors
- Actors define:
  - functionality of components
- Hierarchy:
  - Code generation at a level of the hierarchy produces a new actor definition

Multiple domains may be used in the same model

Software Basis

Build on:

- First version on Titanium (UC Berkeley)
- Second version on Soot (McGill)

Targeting:

- Simulation acceleration
- Embedded software synthesis
- Configurable hardware synthesis

Our Generator Approach

- Actor libraries are built and maintained in Java
  - more maintainable, easier to write
  - polymorphic libraries are rich and small
- Java + Domain translates to target language
  - concurrent and imperative semantics
- Efficiency gotten through code transformations
  - specialization of polymorphic types
  - code substitution using domain semantics
  - removal of unnecessary code

Code transformations (data types)

```
// Original actor source
Token t1 = in.get(0);
Token t2 = in.get(1);
out.send(0, t1.multiply(t2));
```

specialization of Token declarations

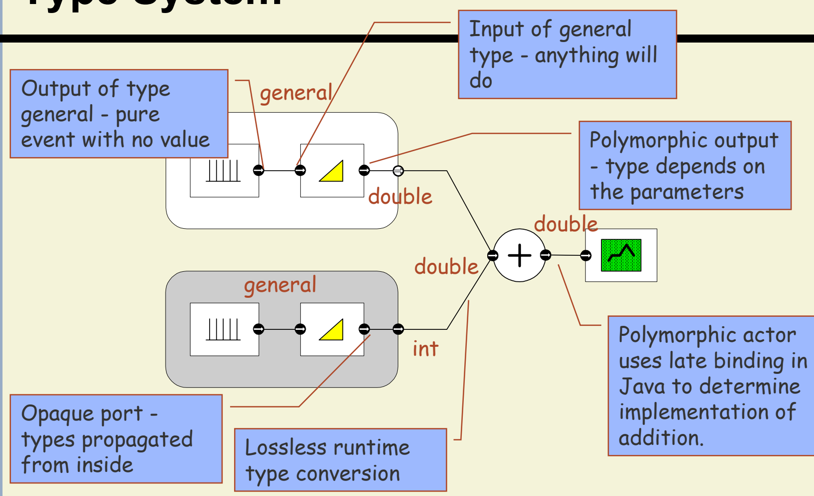
```
// With specialized types
IntMatrixToken t1 = in.get(0);
IntMatrixToken t2 = in.get(1);
out.send(0, t1.multiply(t2));
```

The Ptolemy II type system supports polymorphic actors with propagating type constraints and static type resolution. The resolved types can be used in optimized generated code.

See Jeff Tsay, *A Code Generation Framework for Ptolemy II*

Type system by Yuhong Xiong

Type System



Type System

- Extensible type lattice
- Unification infrastructure
  - Finds a least fixed point
- Composite types
  - records, arrays, matrices
- Higher-order types planned
- Experiments with dependent types
  - encoding domain constraints

Code transformations (domains)

```
// With specialized types
IntMatrixToken t1 = in.get(0);
IntMatrixToken t2 = in.get(1);
out.send(0, t1.multiply(t2));
```

Domain-polymorphic code is replaced with specialized code.

transformation using domain semantics

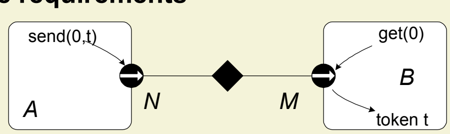
```
// Extended Java with specialized communication
int[][] t1 = _inbuf[0][_inOffset = (_inOffset+1)%5];
int[][] t2 = _inbuf[1][_inOffset = (_inOffset+1)%5];
_outbuf[_outOffset = (_outOffset+1)%8] = t1 * t2;
```

See Jeff Tsay, *A Code Generation Framework for Ptolemy II*

Synchronous Dataflow (SDF) Domain

- **Balance equations (one for each channel):**

  $F_A N = F_B M$
- **Scheduled statically**
- **Decidable resource requirements**



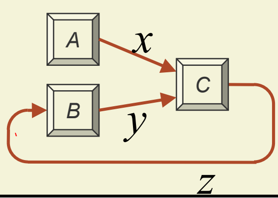
Available optimizations:

- **eliminate checks for input data**
- **statically allocate communication buffers**
- **statically sequence actor invocations (and inline?)**

Synchronous/Reactive Domain

- A discrete model of time progresses as a sequence of "ticks." At a tick, the signals are defined by a fixed point equation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f_{A,t}(1) \\ f_{B,t}(z) \\ f_{C,t}(x,y) \end{bmatrix}$$



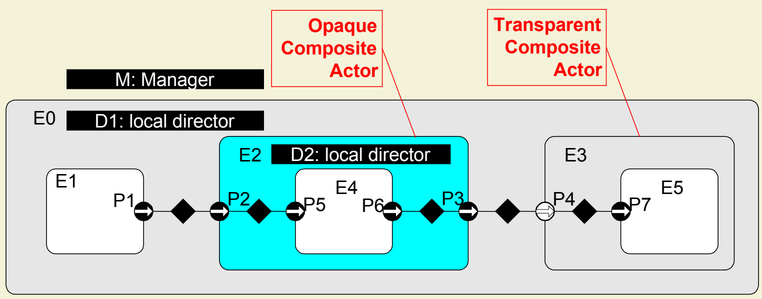
Available optimizations:
• Statically sequence fixed-point iteration
• Communication via registers

Other Domains with Useful Properties for Code Generation

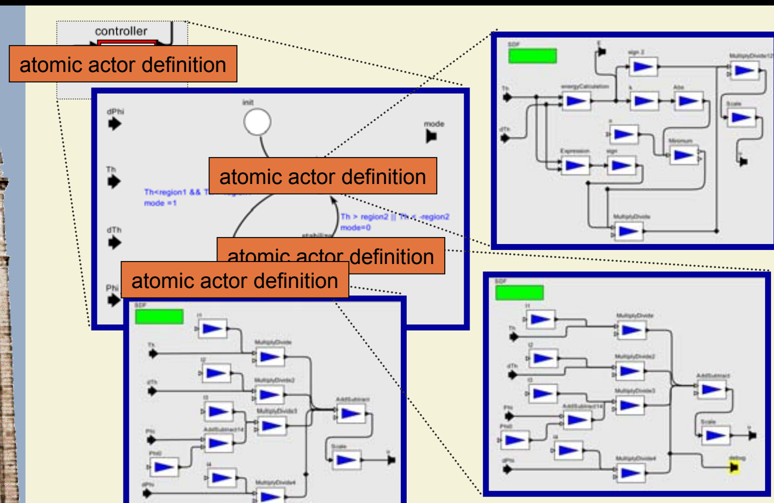
- Strong static analyzability
  - Giotto (time triggered)
  - Finite state machines
  - Discrete time
- Good for hardware descriptions
  - Discrete events
  - Process networks
  - Continuous time (analog hardware)

Hierarchical Heterogeneity

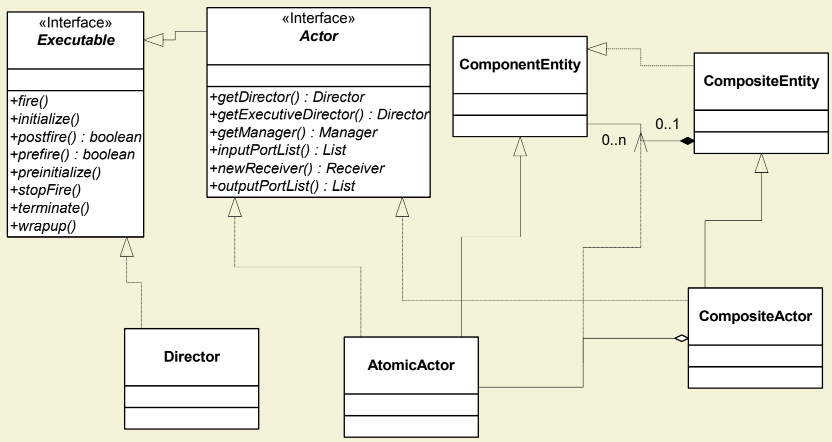
Ptolemy II composes domains hierarchically, where components in a model can be refined into subcomponents where the component interactions follow distinct semantics.



Hierarchical Code Generation



Basic Object Model for Executable Components



Abstract Semantics – How Components Interact

flow of control

- Initialization
- Execution
- Finalization

communication

- Structure of signals
- Send/receive protocols

Abstract Semantics – How Components Interact

flow of control

- **Initialization**
- Execution
- Finalization

communication

- Structure of signals
- Send/receive protocols

- preinitialize()
  - declare static information, like type constraints, scheduling properties, temporal properties, structural elaboration
- initialize()
  - initialize variables

Abstract Semantics – How Components Interact

flow of control

- Initialization
- **Execution**
- Finalization

communication

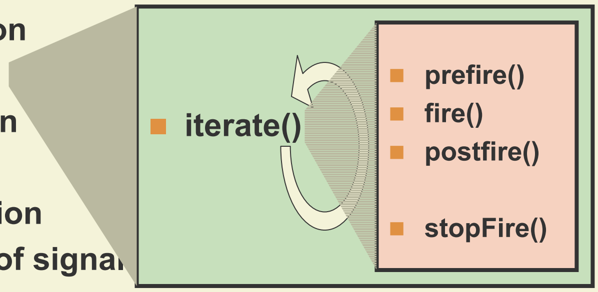
- Structure of signals
- Send/receive protocols

- iterate()

Abstract Semantics – How Components Interact

flow of control

- Initialization
- Execution
- Finalization



communication

- Structure of signa
- Send/receive protocols

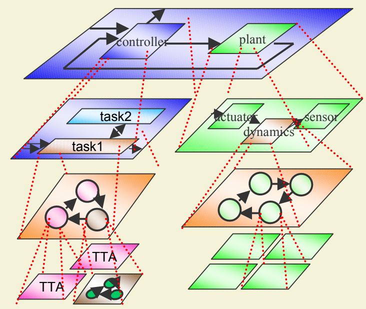
The Key Action Methods

- Prefire()
  - obtain required resources
  - may read inputs
  - may start computations
  - returns a boolean indicating readiness
- Fire()
  - produces results
- Postfire()
  - commits state updates (transactional)
- StopFire()
  - request premature termination

All of these are atomic (non-preemptible)

Benefits

- Composable semantics
  - arbitrarily deep hierarchies
  - heterogeneous hierarchies



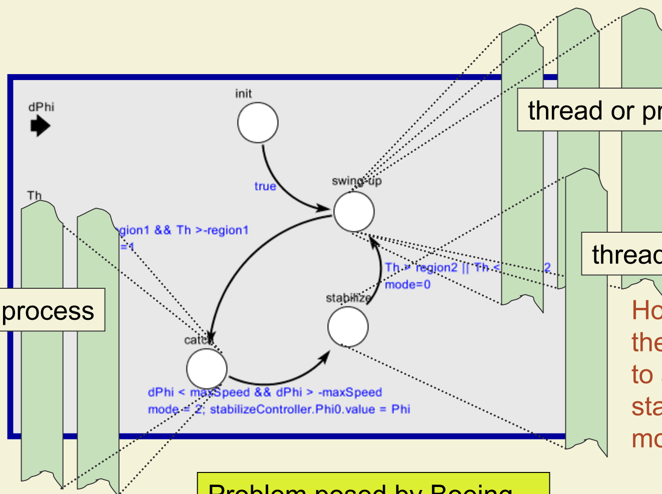
Hierarchical, heterogeneous, system-level model

This Abstract Semantics has Worked For

- Continuous-time models } Hybrid systems
- Finite state machines } Hybrid systems
- Dataflow
- Discrete-event systems
- Synchronous/reactive systems
- Time-driven models (Giotto)
- …

We can even make it work for priority-driven multitasking (RTOS style)!

A Twist: Threaded Models
The Precise Mode Change Problem



How do you get the processes to a quiescent state to take a mode change?

Problem posed by Boeing

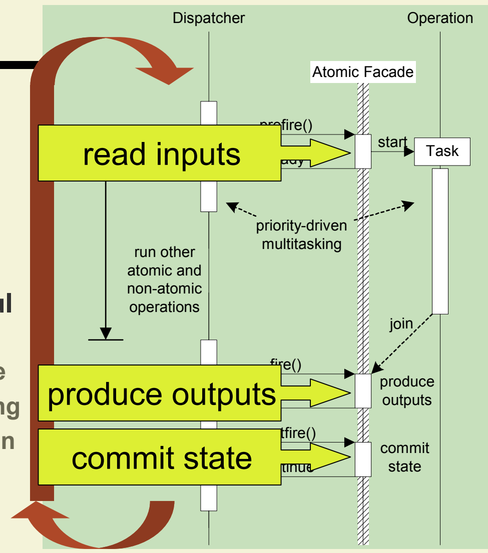
Domain by Jie Liu

HPM Domain
Hierarchical Preemptive Multitasking

- **Objective:**
  - support priority-driven preemptive scheduling
  - use atomic execution, to get composability
  - solve the precise mode change problem
  - make behavior (more) deterministic
- **Solution:**
  - Atomic execution when possible
  - Façade to long-running processes when not
    - Split phase execution (read phase, write phase)

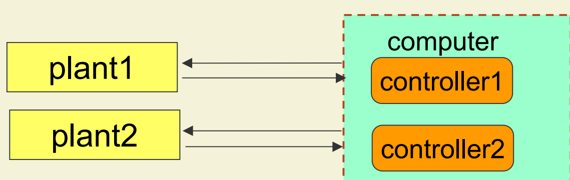
Atomic Façade to Long-Running Computations

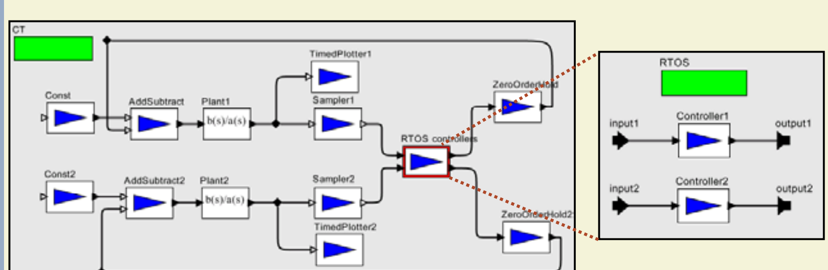
- Each component defines the interaction between the atomic façade and the long-running process.
- There are several useful patterns:
  - allow task to complete
  - enforce declared timing
  - "anytime" computation
  - transactional



Model by Jie Liu

Inter-domain example: Shared-resource controllers





Conclusion

Systematic, ***principled***, ***real-time***, ***heterogeneous***, **hierarchical composition of:**

- **Processes and/or threads**
- **Finite automata (mode controllers)**
- **Other models of computation**
  - **Continuous-time models**
  - **Dataflow models**
  - **…**
- **Code generation**

The key is the abstract semantics of Ptolemy II, which defines hierarchical heterogeneous composition of models of computation.

http://ptolemy.eecs.berkeley.edu