Advanced Tool Architectures

Edited and Presented by
Edward A. Lee, Co-PI
UC Berkeley

Chess Review
November 18, 2004
Berkeley, CA

Tool Projects

- Concurrent model-based design
  - E machine & S machine (Henzinger)
  - Giotto (Henzinger)
  - NP-Click (Keutzer)
  - Metropolis (Sangiovanni-Vincentelli)
  - Ptolemy II (Lee)
  - Streambit (Bodik)
- Meta modeling
  - GME (Sztipanovits, *Vanderbilt*)
  - GREAT=Language,Engine,*C/G*,Debugger (Karsai, *Vanderbilt*)
  - MOF-based Metamodeling (Sztipanovits, *Vanderbilt*)
  - DESERT - Design Space Exploration Tool (Karsai, *Vanderbilt*)
  - UDM - Universal Data Model (Karsai, *Vanderbilt*)
- Verification
  - Blast (Henzinger)
  - CCured (Necula)
  - Chic (Henzinger)
  - SMoLES (Karsai, *Vanderbilt*)

investigator in charge

Tool Building vs. Architecture Principles

- Bottom up: We build tools and applications to make principles concrete and to develop deeper understanding of methods and problems.
- Top down: We identify guiding principles such as meta modeling, abstract syntax, and abstract semantics.

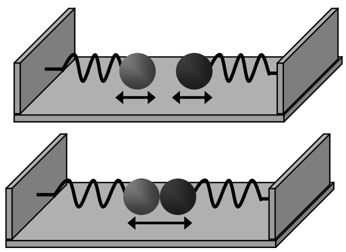
Outline
Separable Tool Architecture Issues

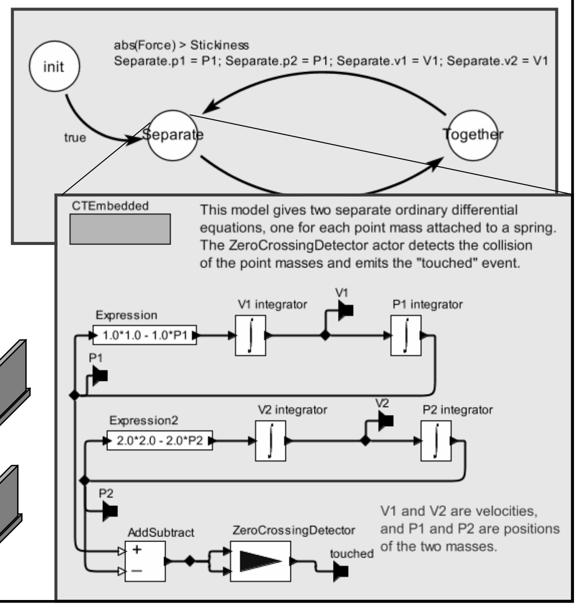
- Abstract Syntax
- Concrete Syntax
- Syntax-Based Static Analysis: Type Systems
- Abstract Semantics
- Concrete Semantics
- Semantics-Based Static Analysis: Verification

Example: HyVisual

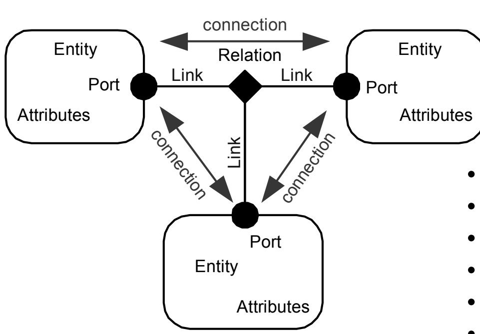
In HyVisual, models of hybrid systems are hierarchical compositions of components that represent state machines and dynamical systems.

What is the underlying structure?





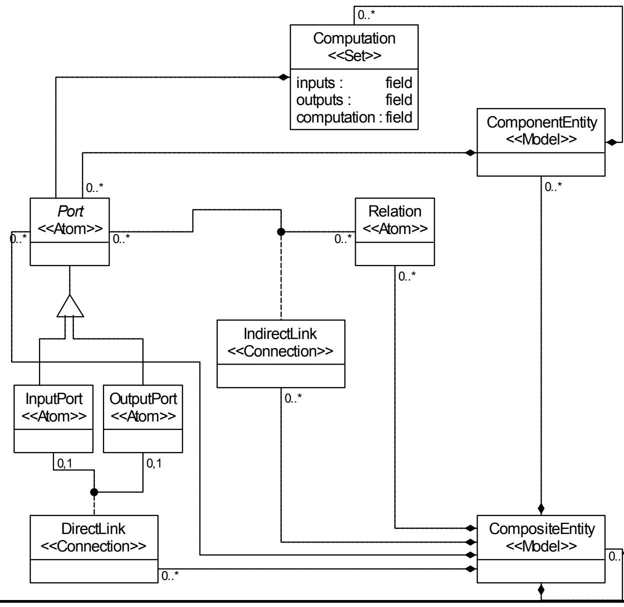
An Abstract Syntax



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

Abstract syntaxes similar to this can be used to describe
- concurrent objects
- interconnected actors
- state machines
- …

Meta-Modeling of an Abstract Syntax



Using GME (from Vanderbilt) an abstract syntax is specified as an object model (in UML) with constraints (in OCL), or alternatively, with MOF.

Such a spec can be used to synthesize visual editors and models transformers.

Meta-model of Ptolemy II abstract syntax, constructed in GME by H. Y. Zheng.

Outline
Separable Tool Architecture Issues

- Abstract Syntax
- Concrete Syntax
- Syntax-Based Static Analysis: Type Systems
- Abstract Semantics
- Concrete Semantics
- Semantics-Based Static Analysis: Verification

Concrete Syntax

Example concrete syntax in XML:

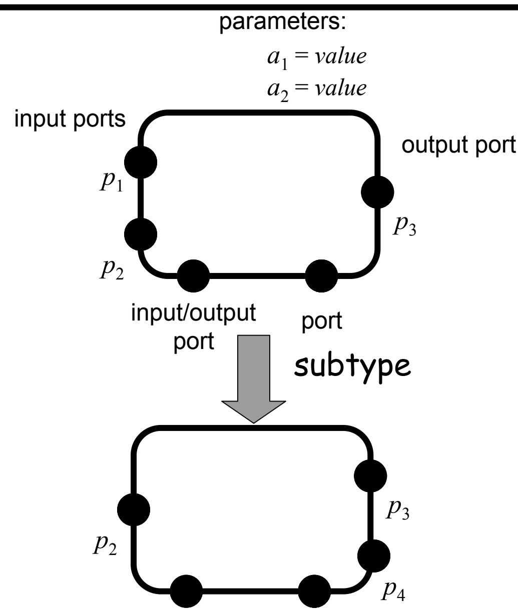
```
...
<entity name="FFT" class="ptolemy.domains.sdf.lib.FFT">
    <property name="order" class="ptolemy.data.expr.Parameter" value="order">
    </property>
    <port name="input" class="ptolemy.domains.sdf.kernel.SDFIOPort">
       ...
    </port>
    ...
</entity>
...
<link port="FFT.input" relation="relation"/>
<link port="AbsoluteValue2.output" relation="relation"/>
...
```

XML and XSLT have made concrete syntax even less important than it used to be. Going a step further, GReAT (from Vanderbilt) works with GME to synthesize model transformers from meta models.

Outline
Separable Tool Architecture Issues

- Abstract Syntax
- Concrete Syntax
- Syntax-Based Static Analysis: Type Systems
- Abstract Semantics
- Concrete Semantics
- Semantics-Based Static Analysis: Verification

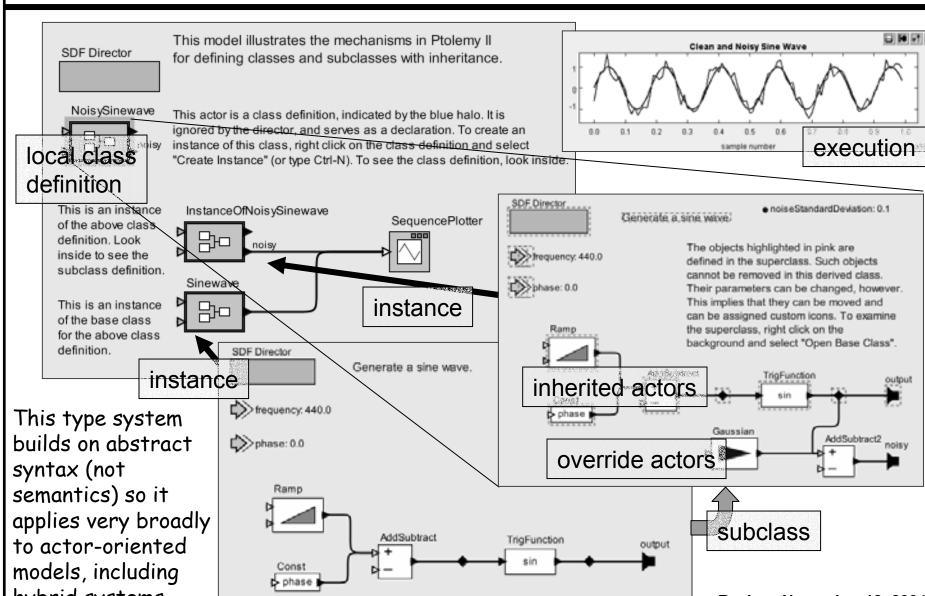
Actor-Oriented Type Systems Interfaces: Ports and Parameters



While types in object-oriented languages are governed by the methods and fields of objects, in actor-oriented languages they are governed by the ports and parameters.

Subtyping needs to be rethought. We have developed an *actor-oriented type system* that depends only an abstract syntax.

Actor-Oriented Type Systems Classes, Subclasses, and Inheritance

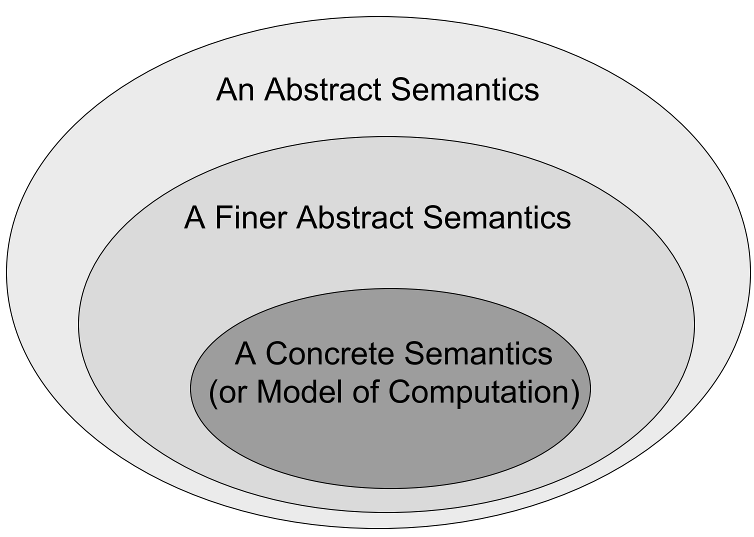


This type system builds on abstract syntax (not semantics) so it applies very broadly to actor-oriented models, including hybrid systems.

Outline
Separable Tool Architecture Issues

- Abstract Syntax
- Concrete Syntax
- Syntax-Based Static Analysis: Type Systems
- Abstract Semantics
- Concrete Semantics
- Semantics-Based Static Analysis: Verification

Where We Are Headed



Tagged Signal Abstract Semantics

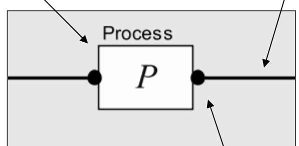
Tagged Signal Abstract Semantics:

a "process" is a subset of the signals with which it interacts.

signal is a member of a set of signals, where the set depends on the model of computation and resolved data type of the connection.

$$P \subset S_1 \times S_2$$

$s_1 \in S_1$



$s_2 \in S_2$

port may be an input or an output, or neither or both. It is irrelevant.

This outlines a general *abstract semantics* that gets specialized. When it becomes concrete you have a *model of computation.*

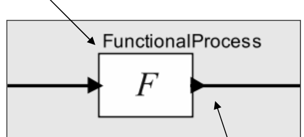
A Finer Abstraction Semantics

Functional Abstract Semantics:

a process is now a function from input signals to output signals.

$$F : S_1 \rightarrow S_2$$

$s_1 \in S_1$



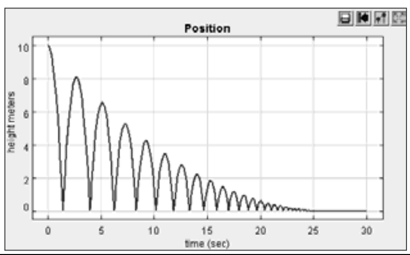
$s_2 \in S_2$

port is now either an input or an output (or both).

This outlines an *abstract semantics* for deterministic producer/consumer actors.

Uses for Such an Abstract Semantics

- Give structure to the sets of signals
  - e.g. Use the Cantor metric to get a metric space.
- Give structure to the functional processes
  - e.g. Contraction maps on the Cantor metric space.
- Develop static analysis techniques
  - e.g. Conditions under which a hybrid systems is provably non-Zeno.



Another Finer Abstract Semantics

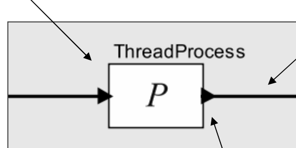
Process Networks Abstract Semantics:

A process is a sequence of operations on its signals where the operations are the associative operation of a *monoid*

sets of signals are *monoids*, which allows us to incrementally construct them. E.g.
- stream
- event sequence
- rendezvous points …

$P \subset S_1 \times S_2$

$s_1 \in S_1$



$s_2 \in S_2$

process is not necessarily functional (can be nondeterministic).

port is either an input or an output or both.

This outlines an abstract semantics for actors constructed as processes that incrementally read and write port data.

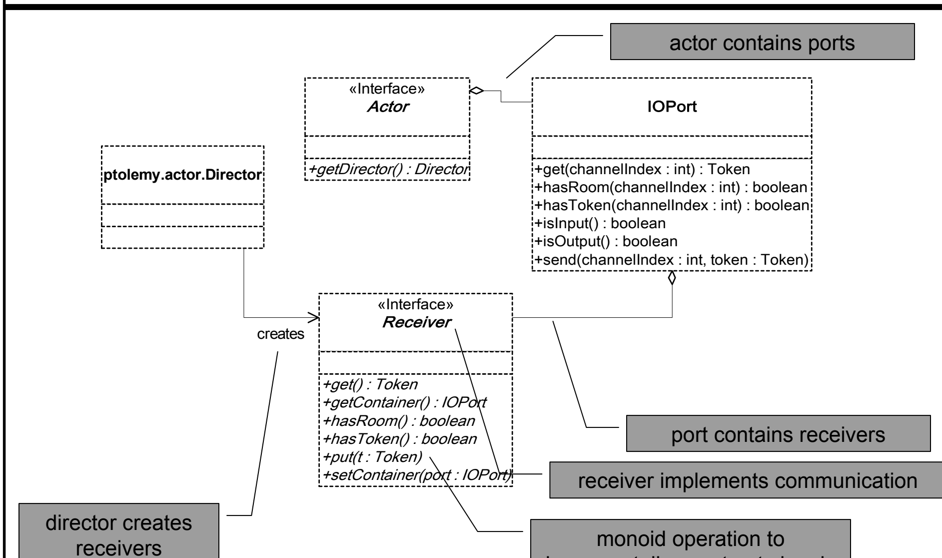
Concrete Semantics that Conform with the Process Networks Abstract Semantics

- Communicating Sequential Processes (CSP) [Hoare]
- Calculus of Concurrent Systems (CCS) [Milner]
- Kahn Process Networks (KPN) [Kahn]
- Nondeterministic extensions of KPN [Various]
- Actors [Hewitt]

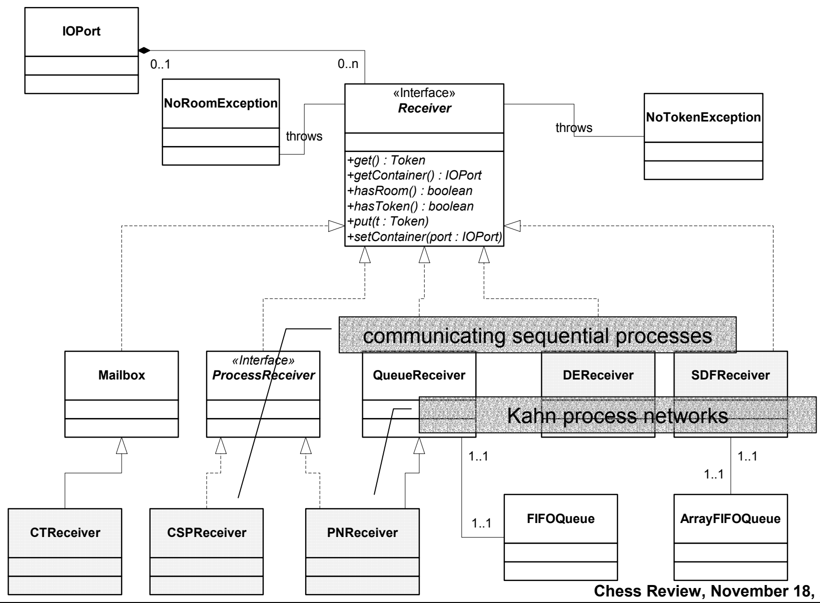
Some Implementations:

- Occam, Lucid, and Ada languages
- Ptolemy Classic and Ptolemy II (PN and CSP domains)
- System C
- Metropolis

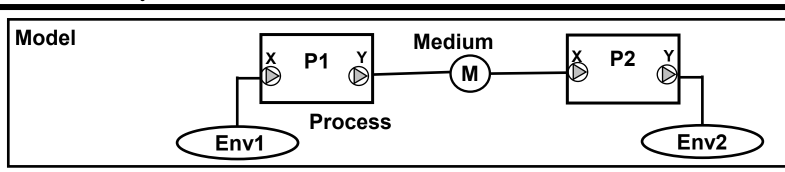
Process Network Abstract Semantics in Ptolemy II



Several Concrete Semantics Refine this Abstract Semantics



Process Network Abstract Semantics in Metropolis




```
process P{
  port reader X;
  port writer Y;
  thread(){
   while(true){
    ...
    z = f(X.read());
    Y.write(z);
   }}}
```

```
interface reader extends Port{
    update int read();
    eval int n();
}
```

```
interface writer extends Port{
    update void write(int i);
    eval int space();
}
```

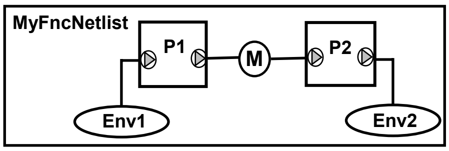
```
medium M implements reader, writer{
   int storage;
   int n, space;
   void write(int z){
       await(space>0; this.writer ; this.writer)
            n=1; space=0; storage=z;
   }
   word read(){ ... }
}
```

Thanks to
Doug Densmore

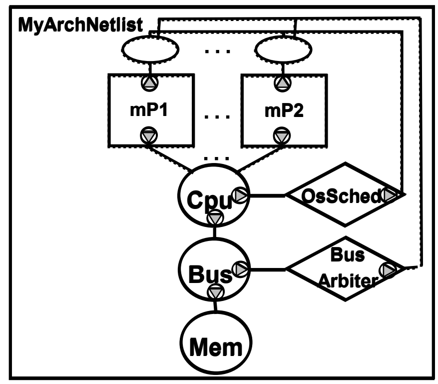
Leveraging Abstract Syntax for Joint Modeling of Architecture and Application

MyMapNetlist

B(P1, M.write) <=> B(mP1, mP1.writeCpu); E(P1, M.write) <=> E(mP1, mP1.writeCpu);
B(P1, P1.f) <=> B(mP1, mP1.mapf); E(P1, P1.f) <=> E(mP1, mP1.mapf);
B(P2, M.read) <=> B(P2, mP2.readCpu); E(P2, M.read) <=> E(mP2, mP2.readCpu);
B(P2, P2.f) <=> B(mP2, mP2.mapf); E(P2, P2.f) <=> E(mP2, mP2.mapf);



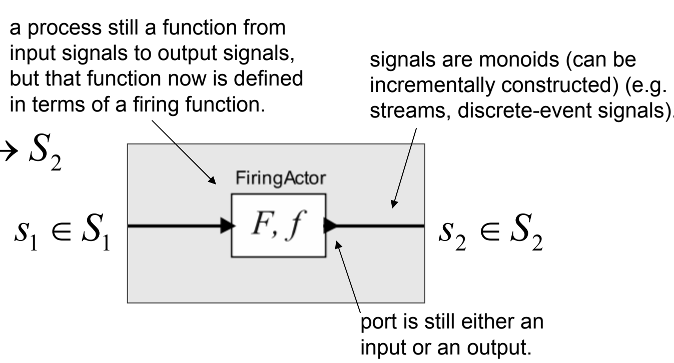
The abstract syntax provides natural points of the execution (where the monoid operations are invoked) that can be synchronized across models. Here, this is used to model operations of an application on a candidate implementation architecture.



A Finer Abstract Semantics

Firing Abstract Semantics:

$F : S_1 \to S_2$



The process function F is the least fixed point of a functional defined in terms of f.

Models of Computation that Conform to the Firing Abstract Semantics

- Dataflow models (all variations)
- Discrete-event models
- Time-driven models (Giotto)

In Ptolemy II, actors written to the *firing abstract semantics* can be used with directors that conform only to the process network abstract semantics.

Such actors are said to be *behaviorally polymorphic*.

Leveraging the Abstract Semantics to get "Schedule Carrying Code" (SCC)

Giotto code

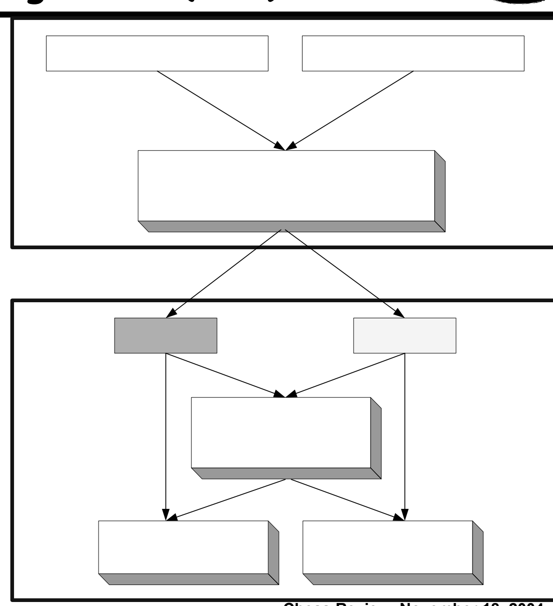
- firings that are concurrent yet atomic
- periodic tasks and drivers
- unit-delay state semantics
- multi-modal

Embedded (E) code

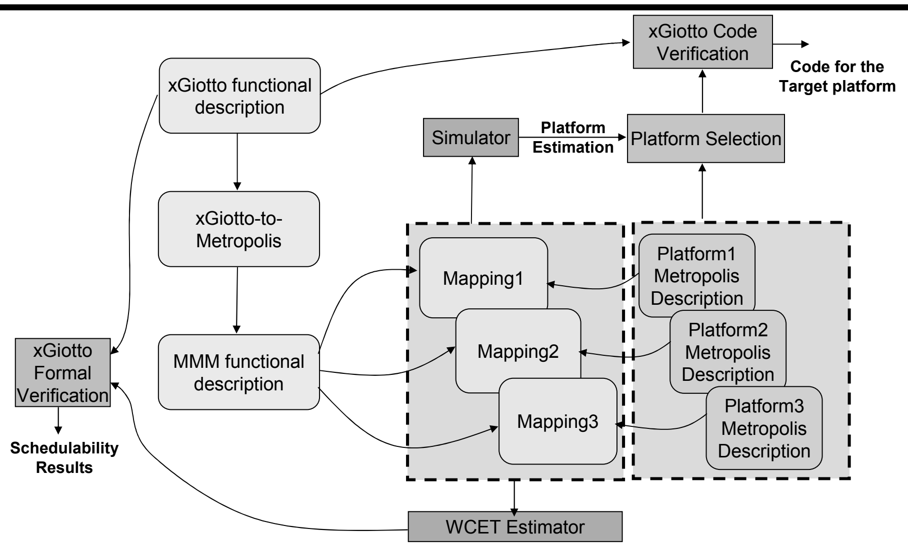
- environment interaction
- task release

Scheduling (S) code

- task execution
- communication schedule



xGiotto and Metropolis



Actor Language for the Firing Abstract Semantics: Cal

Cal is an experimental actor language designed to provide statically inferable actor properties w.r.t. the firing abstract semantics. E.g.:

```
actor Select () S, A, B ==> Output:

    action S: [sel], A: [v] ==> [v]
    guard sel end

    action S: [sel], B: [v] ==> [v]
    guard not sel end
end
```

Inferable firing rules and firing functions:

$$U_1 = \{\langle (\text{true}), (v), \bot \rangle : v \in \mathbf{Z}\}, f_1 : \langle (\text{true}), (v), \bot \rangle \mapsto (v)$$
$$U_2 = \{\langle (\text{false}), \bot, (v) \rangle : v \in \mathbf{Z}\}, f_2 : \langle (\text{false}), \bot, (v) \rangle \mapsto (v)$$

Thanks to Jorn Janneck, Xilinx

A Still Finer Abstract Semantics

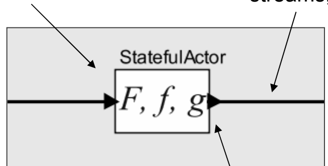
Stateful Firing Abstract Semantics:

a process still a function from input signals to output signals, but that function now is defined in terms of two functions.

signals are monoids (can be incrementally constructed) (e.g. streams, discrete-event signals).

$F : S_1 \to S_2$

$s_1 \in S_1$



$s_2 \in S_2$

$f : S_1 \times \Sigma \to S_2$

$g : S_1 \times \Sigma \to \Sigma$

state space

port is still either an input or an output.

The function f gives outputs in terms of inputs and the current state. The function g updates the state.

Models of Computation that Conform to the Stateful Firing Abstract Semantics

- Synchronous reactive
- Continuous time
- Hybrid systems

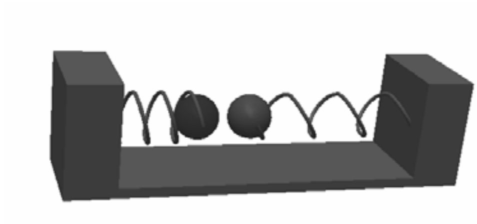
Stateful firing supports iteration to a fixed point, which is required for hybrid systems modeling.

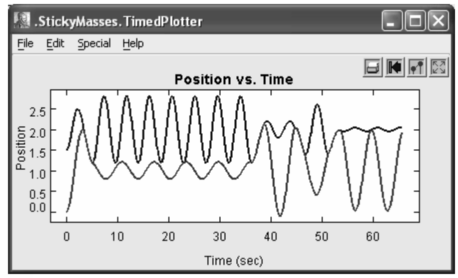
In Ptolemy II, actors written to the stateful firing abstract semantics can be used with directors that conform only to the firing abstract semantics or to the process network abstract semantics.

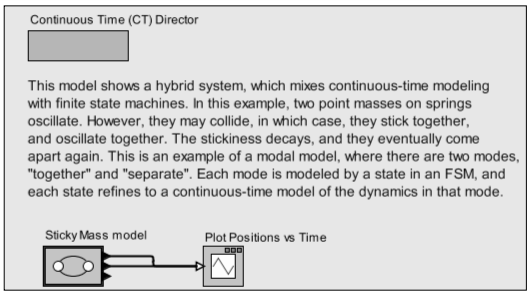
Such actors are said to be *behaviorally polymorphic*.

Leveraging This Abstract Semantics in HyVisual (based on Ptolemy II)

Masses on Springs

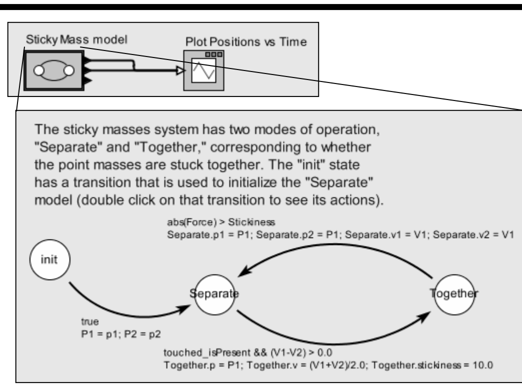






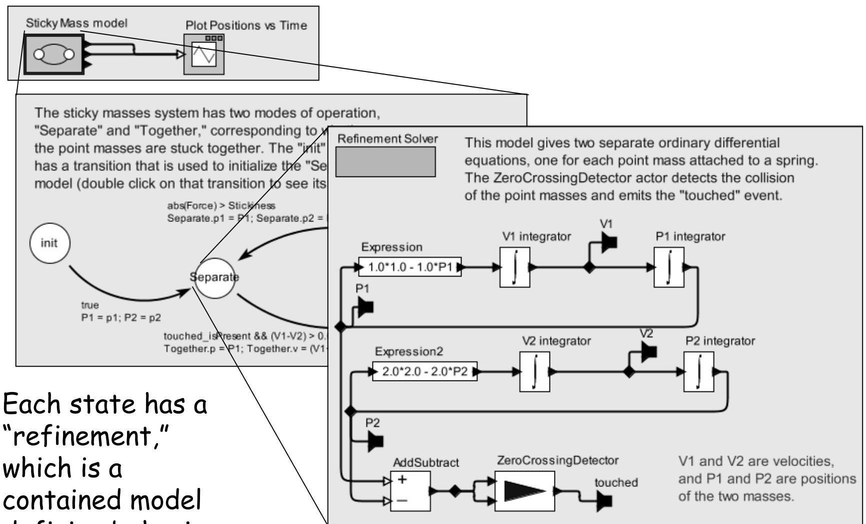
Consider two masses on springs which, when they collide, will stick together with a decaying stickiness until the force of the springs pulls them apart again.

Structure of the Spring-Masses Model



A component in the continuous-time top-level model is defined by a finite state machine. The continuous time model requires the stateful firing abstract semantics for the ODE solver to work properly across these levels of the hierarchy.

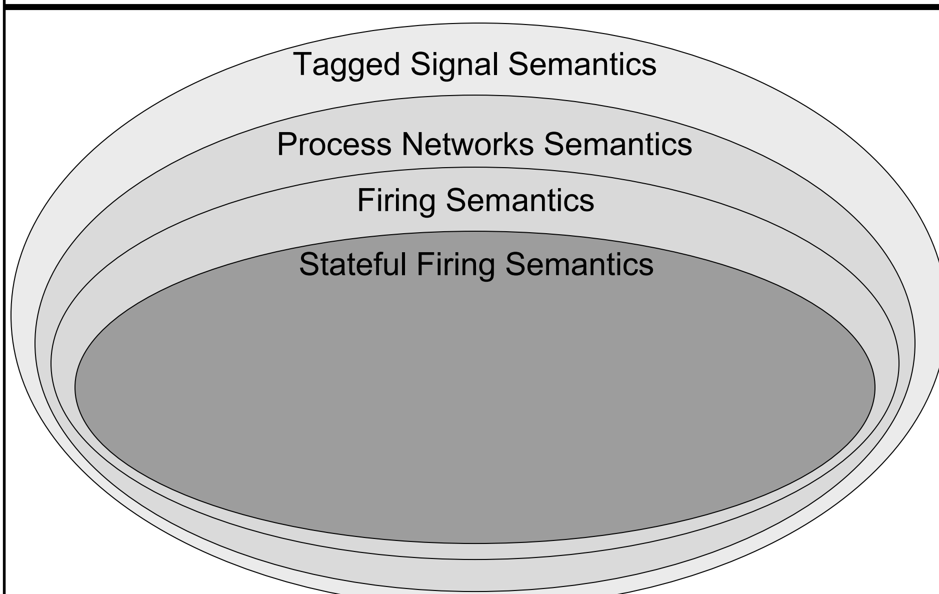
Structure of the Spring-Masses Model



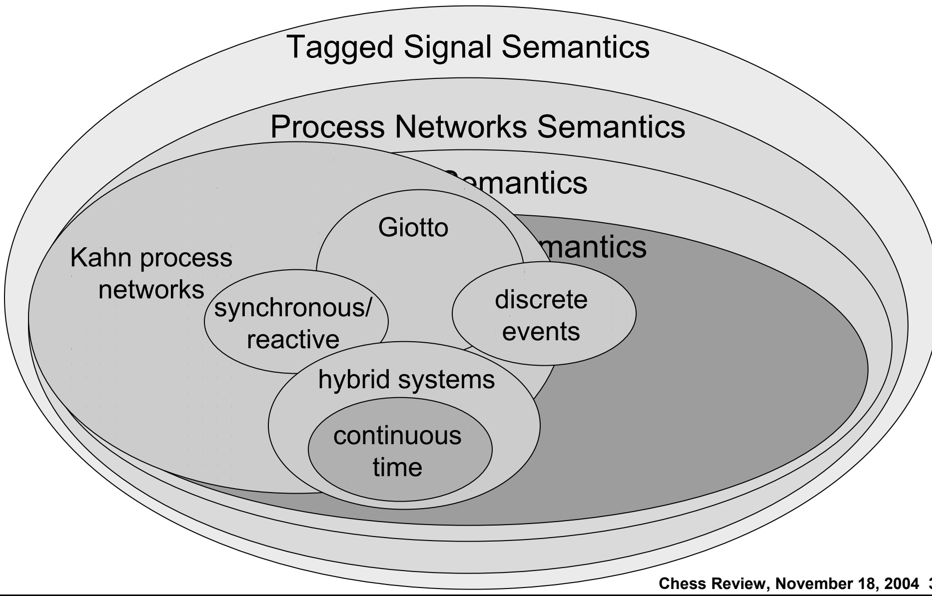
Each state has a "refinement," which is a contained model defining behavior.

This requires a composable abstract semantics.

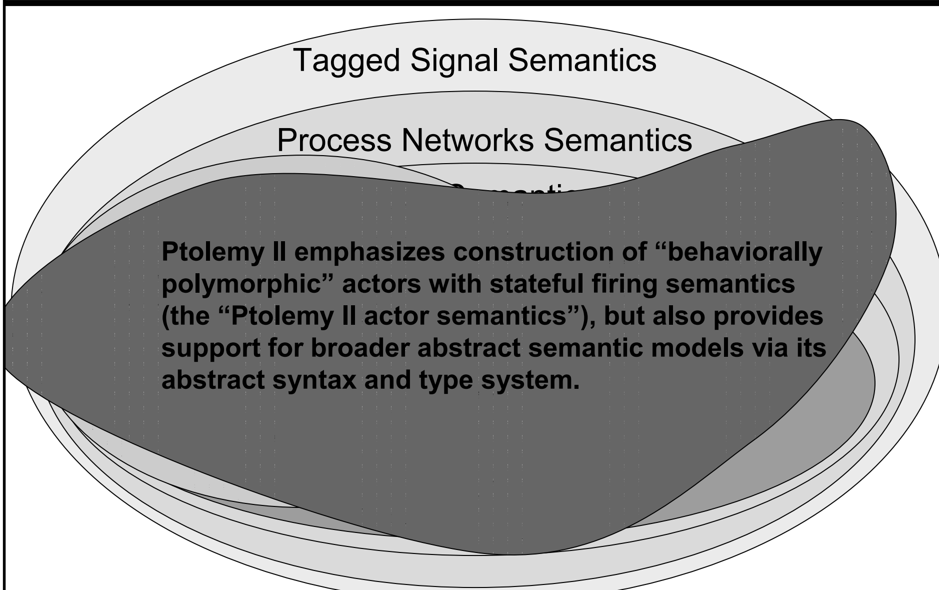
Where We Are



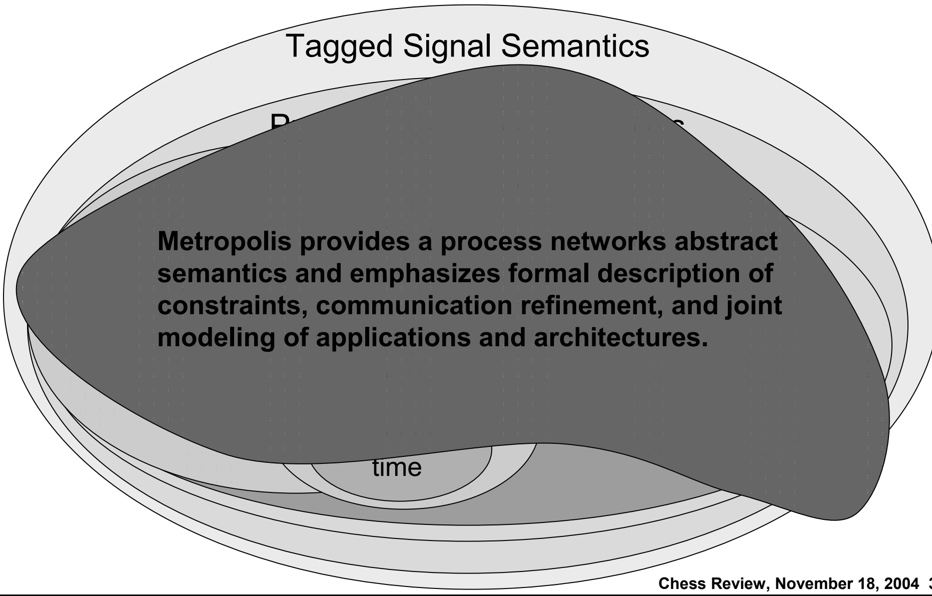
Where We Are



Meta Frameworks: Ptolemy II



Meta Frameworks: Metropolis



Outline
Separable Tool Architecture Issues

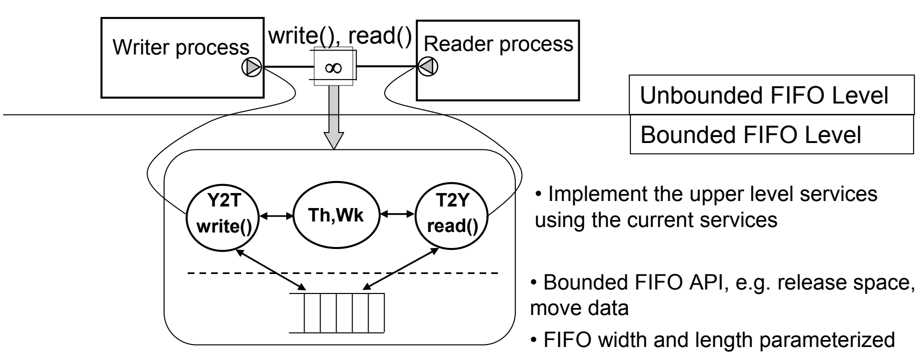
- Abstract Syntax
- Concrete Syntax
- Syntax-Based Static Analysis: Type Systems
- Abstract Semantics
- Concrete Semantics
- Semantics-Based Static Analysis: Verification

Verification
Semantics-Based Static Analysis

- Refinement verification in Metropolis
- CHIC model checker for interface checking
- CHIC integration with Ptolemy II
- Blast

Leveraging the Abstract Semantics for Refinement Verification in Metropolis

Example: a unbounded FIFO v.s. a bounded FIFO with the finer service.



- Implement the upper level services using the current services
- Bounded FIFO API, e.g. release space, move data
- FIFO width and length parameterized

: refinement relation

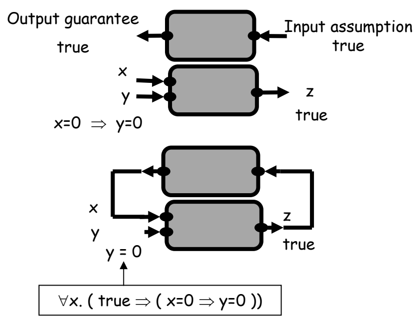
- Metropolis represent both levels of abstraction explicitly, rather than replacing the upper level.
- Refinement relation is associated with properties to preserve through the refinement.

Thanks to Doug Densmore

Chic: A Tool for Checking Interface Compatibility

(Thomas A. Henzinger et. al.)

Interface: Expresses assumptions made by module about environment, and guarantees made by module if assumptions are satisfied.
Interface = Behavioral type



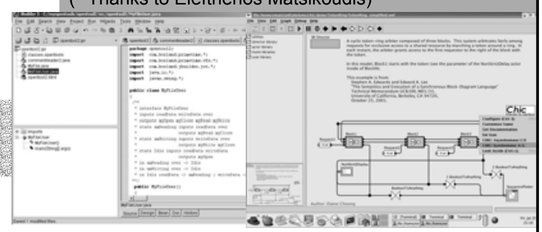
Compatibility checking is a game between System and Environment; winning strategy of Environment gives correct way to use System.

Software Module interfaces allow pushdown analysis to check safety properties of recursive software components.

Resource interfaces: automata-based type system for compositional resource-aware analysis of embedded software. eg. Node Limit Interfaces express requirements like mutex, limited buffer size, limited peak power. Path Limit Interfaces express requirements like limited battery capacity. Compositional and scalable.

Web Service interfaces allow checking temporal properties of interaction between service components.

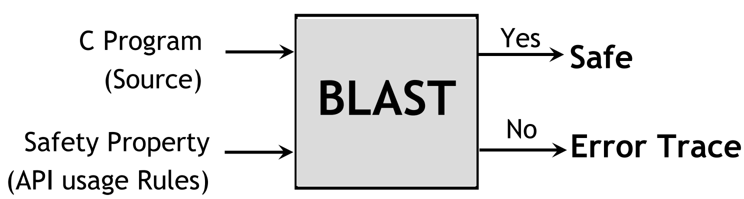
Chic 1.1 is available as a plug-in for JBuilder, Ptolemy*. Implemented in Java. Supports static, dynamic (including pushdown) and resource interfaces. Support for web service interfaces is under development.
(* Thanks to Eleftherios Matsikoudis)



Download Chic 1.1 today !! http://www.eecs.berkeley.edu/~tah/Chic/

BLAST

Berkeley Lazy Abstraction Software Verification Tool



- Automatic counterexample-guided abstraction-refinement
- Scales to 100Kloc

The Big Question: How to Give Semantic Meta Models that are Usefully Manipulable

Key ideas guiding us:

- Abstract semantics
- Ptolemy II directors
- Metropolis quantity managers
- The Metropolis language of constraints
- Interface theories
- Behavioral type systems
- Temporal logics (e.g. TLA)
- Set-valued semantics
- …