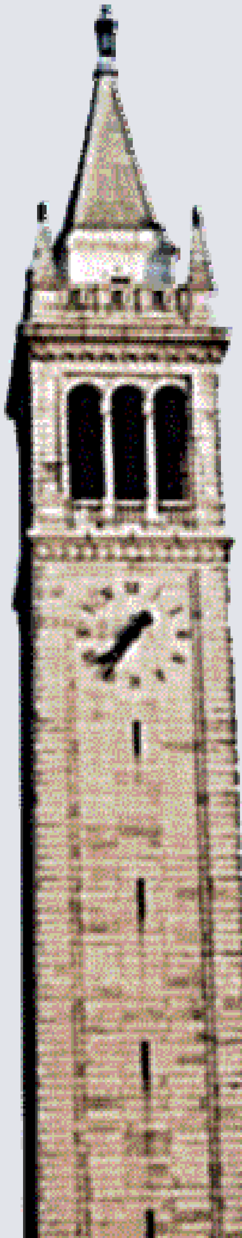
Advanced Tool Architectures

Edited and presented by
Edward A. Lee
UC Berkeley

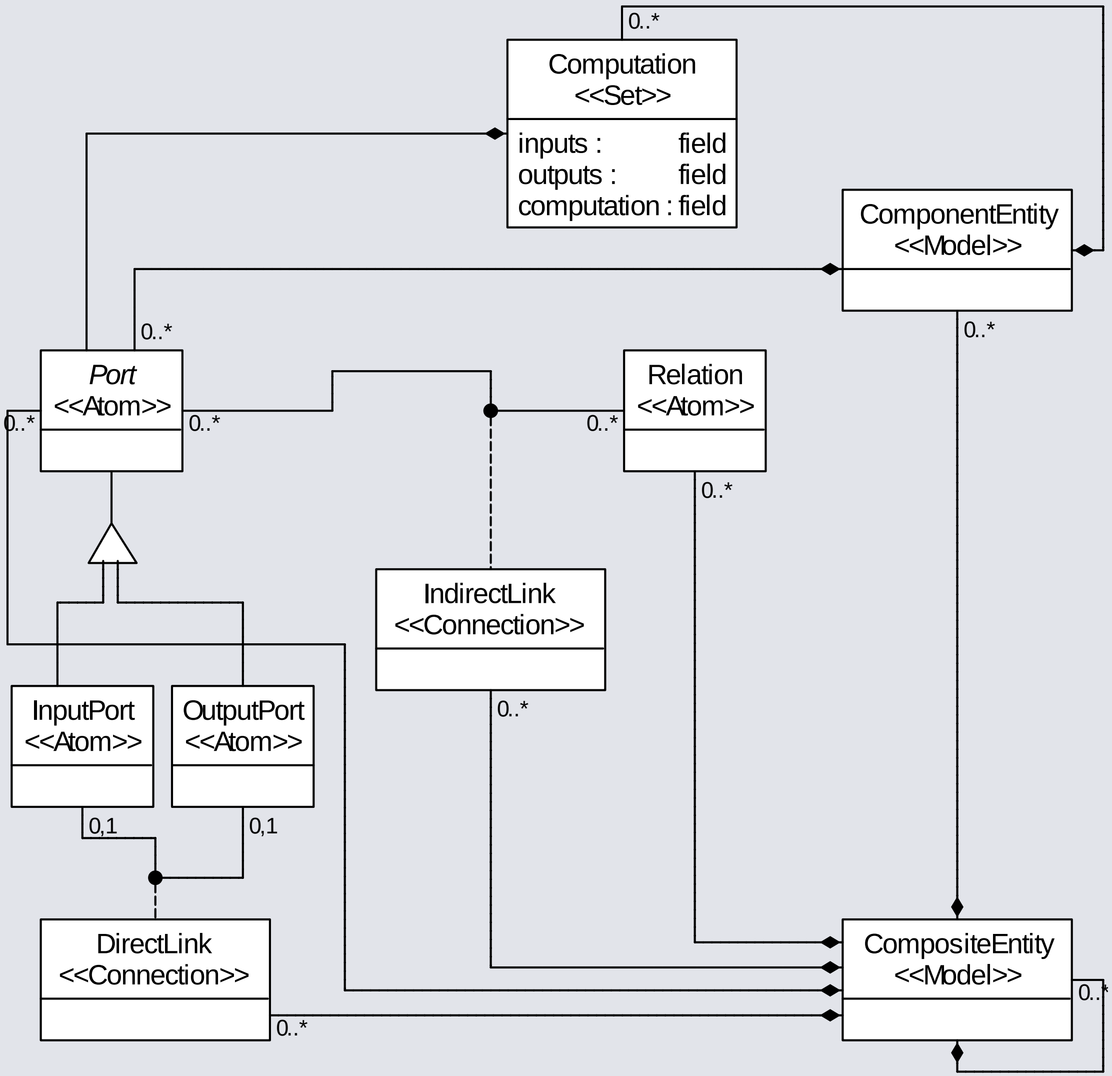


Chess Review
October 4, 2006
Alexandria, VA

Thrust III: Advanced Tool Architectures

- Syntax and Synthesis
  - Semantic Composition
  - Visual Concrete Syntaxes
  - Modal Models
- Interface Theories
- Virtual Machine Architectures
- Components for Embedded Systems

Major Concepts 1: Metamodeling of 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.

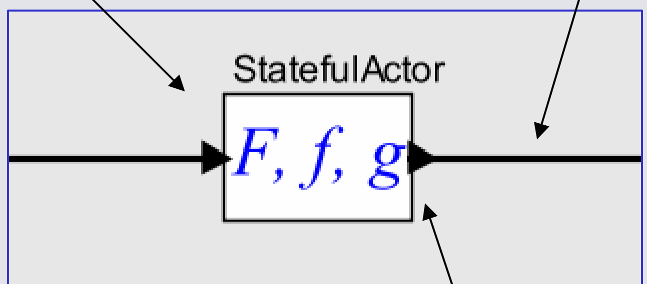
Major Concepts 2:
Actor Abstract Semantics

A process is a function from input signals to output signals. That function is defined in terms of two functions.

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

$$F : S_1 \to S_2$$

$$s_1 \in S_1$$



$$s_2 \in S_2$$

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

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

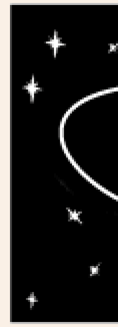
state space

A port is 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.

Major Concepts 3: Experimental Tools & Frameworks

HyVisual - Hybrid System Visual Modeler

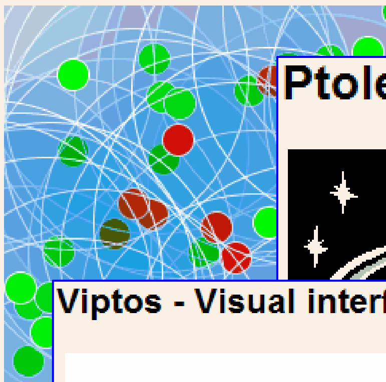


Metropolis

Metropolis: Design Environment for Heterogeneous Systems

VisualSense

Visual editor and simulator for wireless sensor network systems



Ptolemy II

Viptos - Visual interface between Ptolemy and TinyOS

Viptos

Universal Data Model (UDM)

The Generic Modeling Environment

GME 5

The Graph Rewrite And Transformation (GReAT) tool suite

What We Have Learned

Hybrid and embedded software systems demand a different approach to computation.

Instead of a Program Specifying…

$$f : \{0,1\}^* \to \{0,1\}^*$$

… A Program Should Specify

$$f : [T \to \{0,1\}^*]^P \to [T \to \{0,1\}^*]^P$$

"actor" f; *"signal"* $[T \to \{0,1\}^*]$; *"signal"* $[T \to \{0,1\}^*]$

…where T is a set representing time, precedence ordering, causality, synchronization, etc.

The Catch…

$$f:[T \to \{0,1\}^*]^P \to [T \to \{0,1\}^*]^P$$

- This is not what (mainstream) programming languages do.
- This is not what (mainstream) software component technologies do.

The second problem is easier to solve…

Actor-Oriented Design

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

Examples of Actor-Oriented "Languages"

- CORBA event service (distributed push-pull)
- ROOM and UML-2 (dataflow, Rational, IBM)
- VHDL, Verilog (discrete events, Cadence, Synopsys, ...)
- LabVIEW (structured dataflow, National Instruments)
- Modelica (continuous-time, constraint-based, Linkoping)
- OPNET (discrete events, Opnet Technologies)
- SDL (process networks)
- Occam (rendezvous)
- Simulink (Continuous-time, The MathWorks)
- SPW (synchronous dataflow, Cadence, CoWare)
- …

Many of these are domain specific.

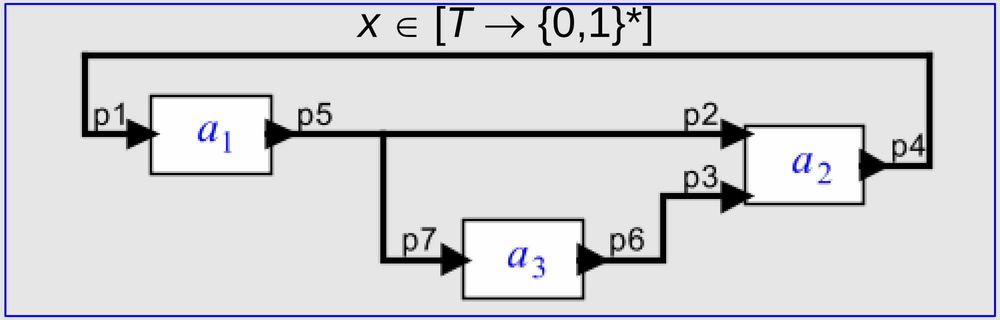
Many of these have visual syntaxes.

The semantics of these differ considerably, but all can be modeled as

$$f : [T \to \{0,1\}^*]^P \to [T \to \{0,1\}^*]^P$$

with appropriate choices of the set T.

Actor-Oriented Component Composition



- Cascade connections
- Parallel connections
- Feedback connections

If actors are functions on signals, then the nontrivial part of this is feedback.

Some of the Possible Models of Computation:

- *Time-Triggered*
- *Discrete Events*
- *Dataflow*
- *Rendezvous*
- *Synchronous/Reactive*
- *Continuous Time*
- *…*

Major Ongoing Efforts

- Model Transformations and Code Generation
- Abstract Semantics and heterogeneous modeling
- Interface algebras for
  - real-time
  - causality
  - refinement
- Scalability
- Models for distributed real-time computing
- Relationships between models of computation
  - Unification of SR/DE/CT semantics
  - Trading latency for composability in time-based models

Focus on Interface Algebras

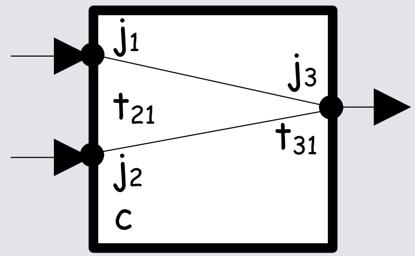
Algebraic interface theories for:

- Real-time
  - [Matic, Henzinger]
- Causality
  - [Lee, Zheng, Zhou]
- Refinement
  - [Passerone, Al Sangiovanni-Vincentelli]

Interface Algebra for Real-Time Graphs

Assumption

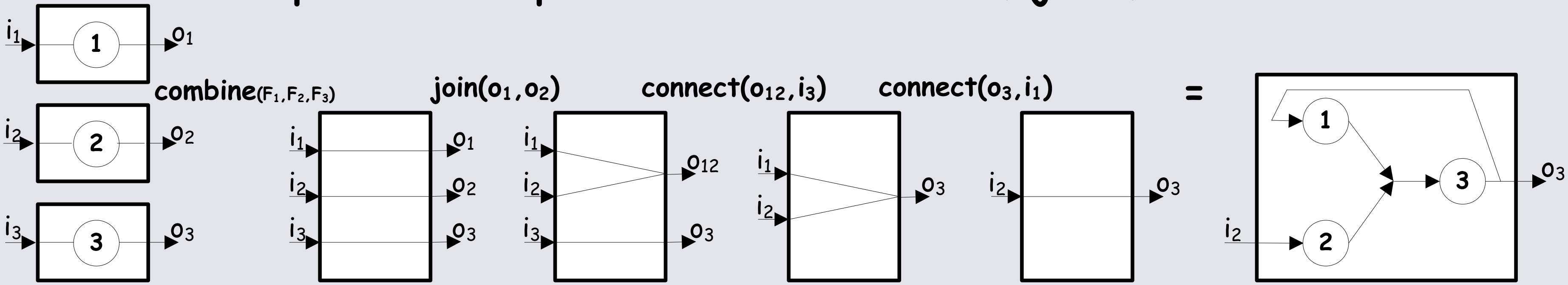
input jitter
offset
input jitter
resource capacity



Guarantee

output jitter
output latency

Composition operations: connect, join, combine

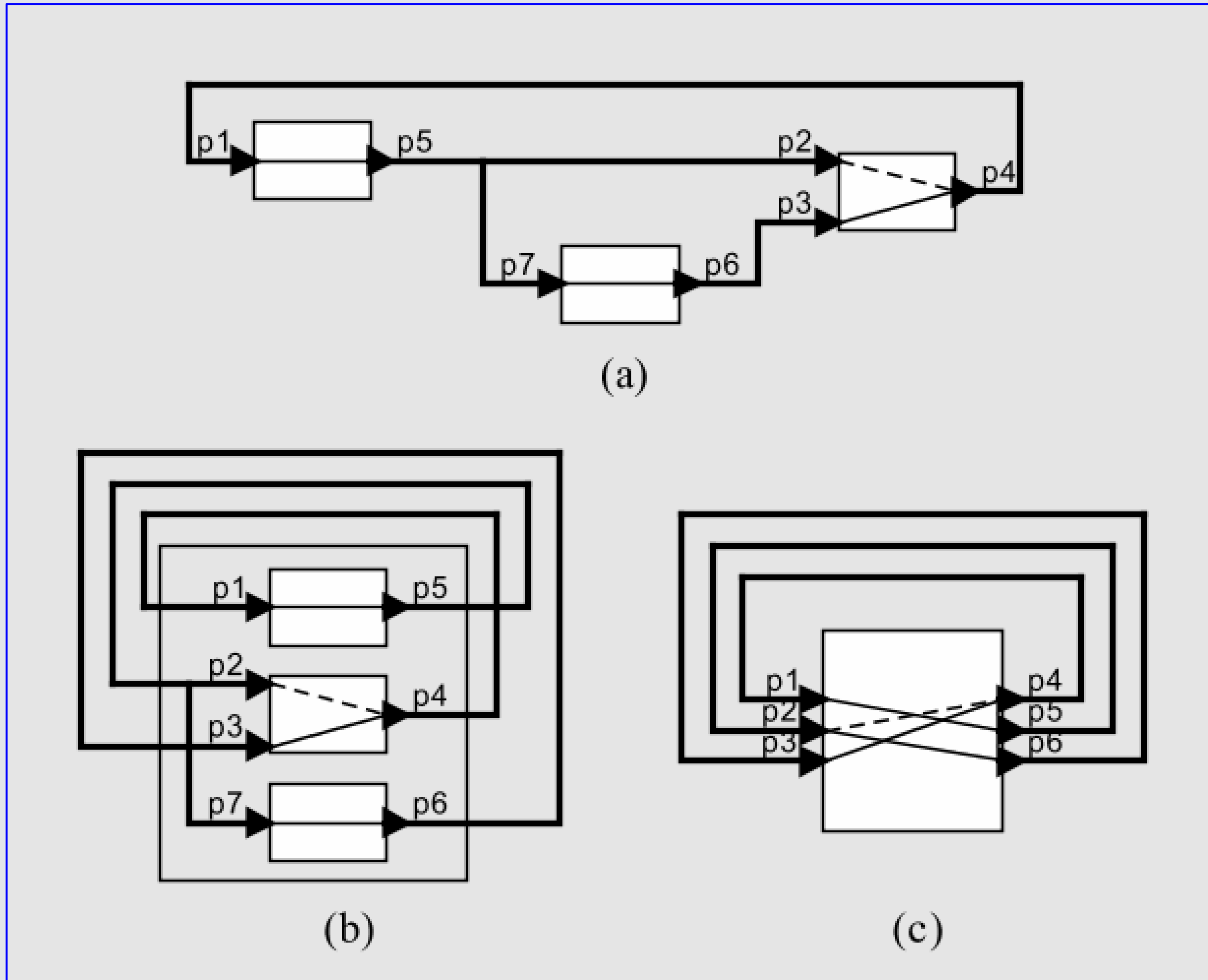


Incremental design

Independent refinement

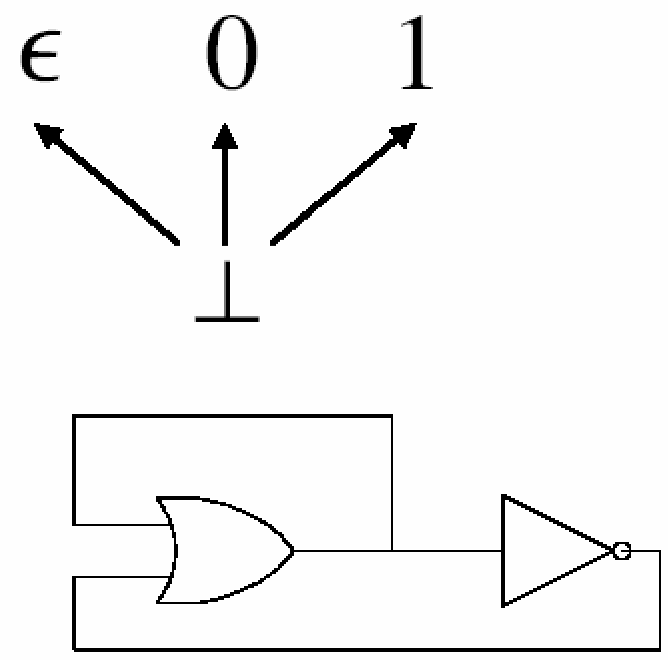
Interface Algebra for Causality Analysis

An algebra of interfaces provides operators for cascade and parallel composition and necessary and sufficient conditions for causality loops, zero-delay loops, and deadlock.



Example: Fixed Point is Not Constructive

In a synchronous language, the program at the right has a unique non-empty behavior, but that behavior cannot be found constructively by repeatedly application of monotonic functions.



(⊥, ⊥) and (1, 0) are fixed point solutions.

Example: Causality Loops

In a synchronous language, the programs at the right do not have unique non-empty behaviors. This defect is called a causality loop.

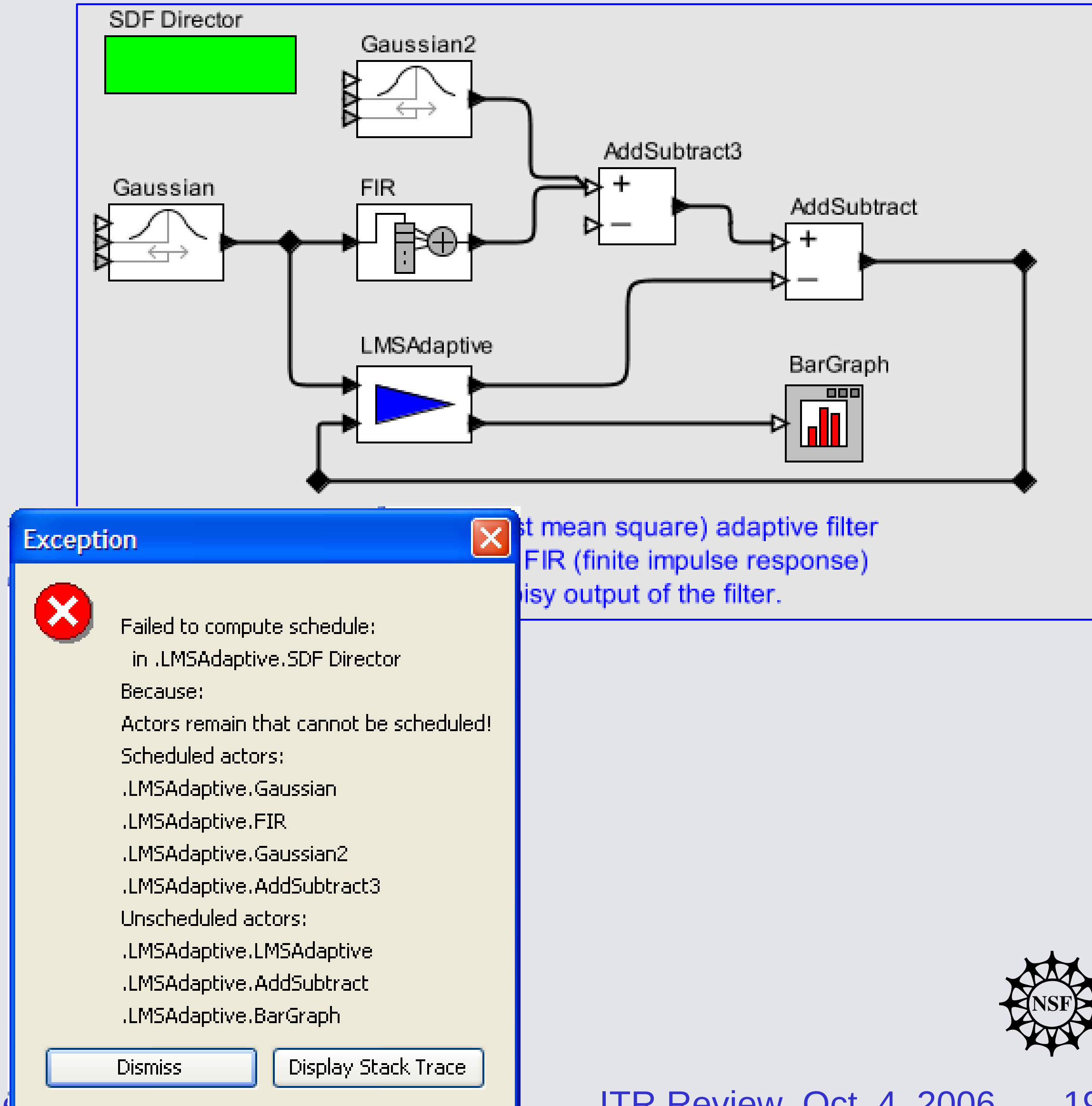
⊥ is the only fixed point solution.

⊥, 0, and 1 are all fixed point solutions.

Example: Deadlock

In a process networks and dataflow models, programs may exhibit deadlock, where behavior is empty or finite.

Deadlock in such programs is, in general, undecidable.



Tools and Software Frameworks

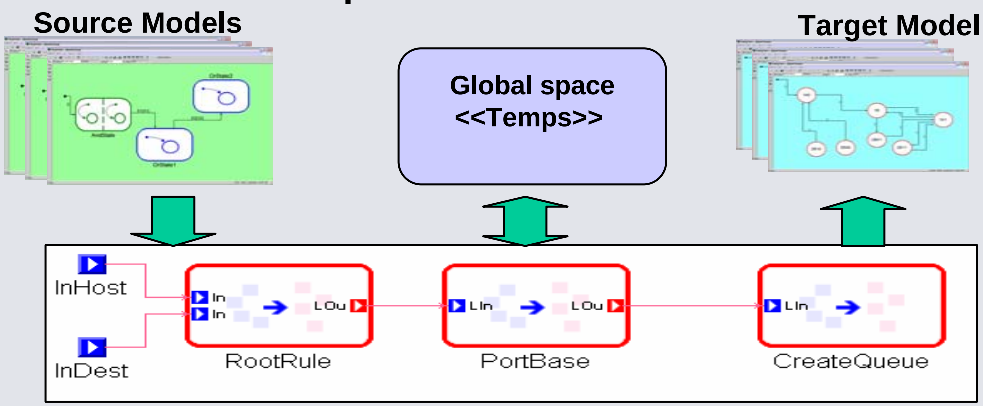
- Software Tools
  - GReAT
  - HyVisual
  - Visualsense
  - Viptos
- Meta Tools
  - GME
  - Metropolis & Metropolis II
  - Ptolemy II
  - UDM

Meta tools are software frameworks that function as laboratories for model-based design

Model Transformation Tools

The Graph Rewrite And Transformation (GReAT) tool suite, Vanderbilt.

Global “spaces” for Transformations

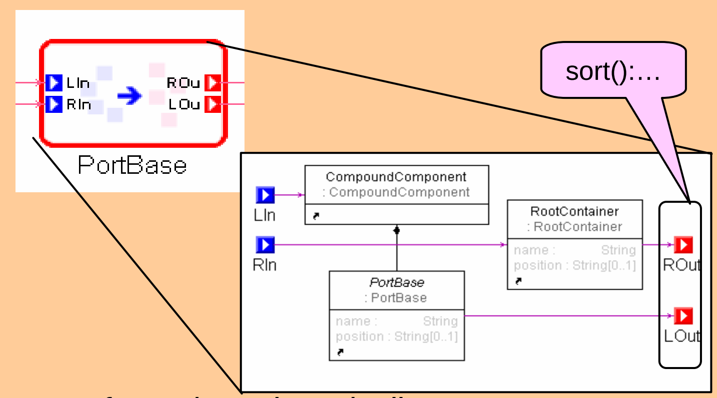


Global spaces hold intermediate results of the transformation
<u>**Consequence**</u>: The transformations are simplified.

Model Transformation Tool features:

- User code libraries
- Integration with new development platform (Microsoft VS 7+)
- Support for XML namespaces
- Integration with Java
- Support for structured text input and output with declarative specification of the syntax

Sorting the transformation results



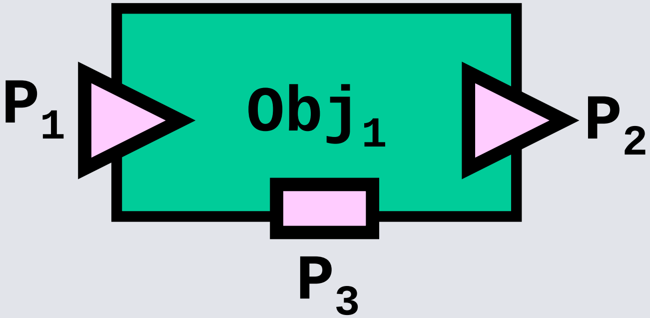
A transformation rule typically operates on a *sequence* of matched objects that could be sorted **after** the rule is applied.
<u>**Consequence**</u>: Model transformation results are ordered by the sorting function.

Additional language features:

- Distinguished cross-product: a new built-in operator of the language that refines pattern matching semantics
- Match-any associations: “wild-card” pattern matching construct for matching arbitrary associations
- Support for automatic connection of multi-ported objects in the modeling tool

Current Directions in Meta Tools
Metropolis II

- An object is the basic entity
- An object interacts with the outside world using *ports*
- A port has an associated set of *services*
- A port can be of three types:
  - Input: Services provided by object
  - Output: Services used by object
  - View: Services that can be observed
- An object may have zero or more input, output and view ports
  - Services can overlap

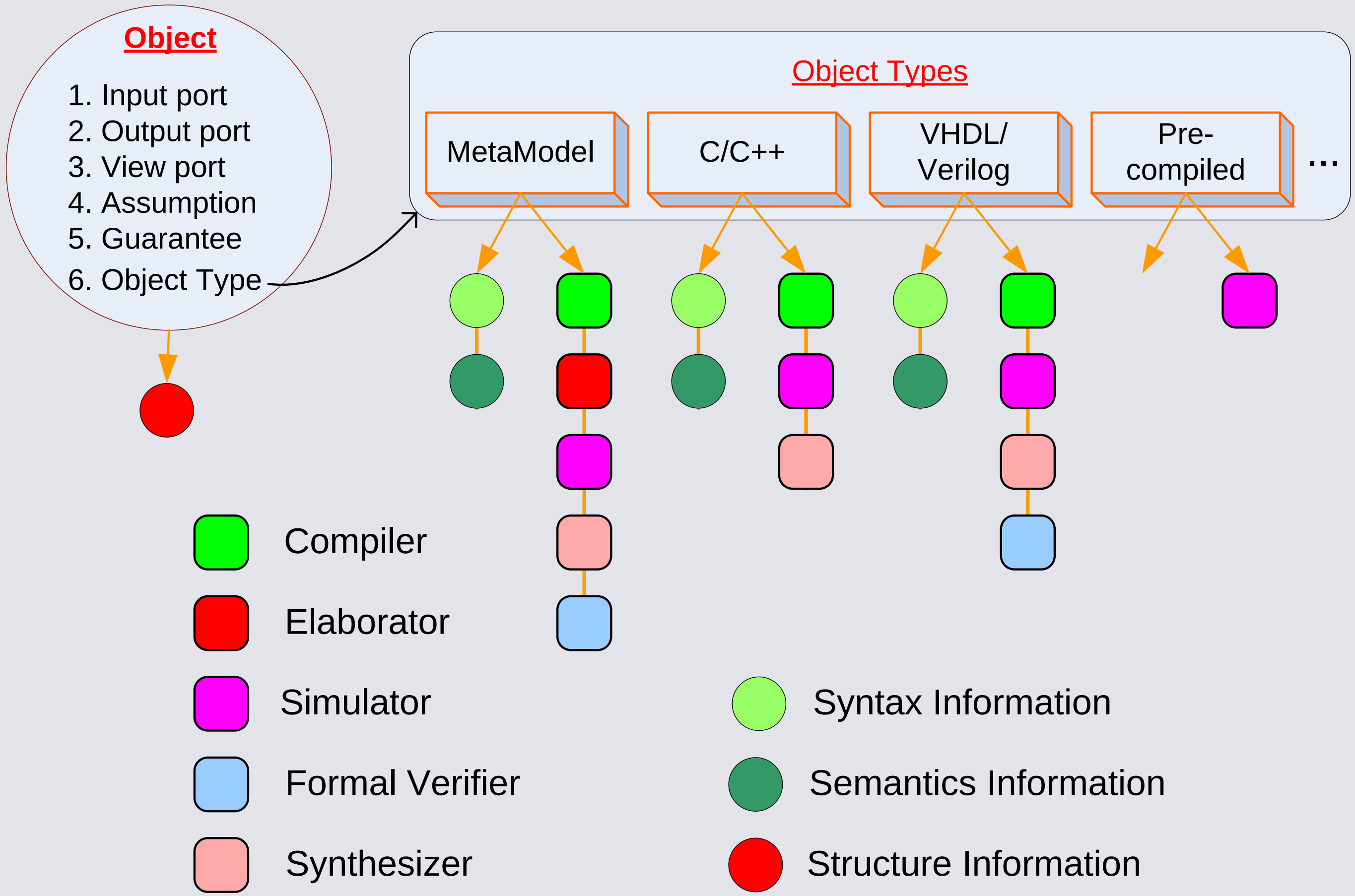


$S(P_1)=\{S_1,S_2,S_3\}$

$S(P_2)=\{S_3,S_4,S_5\}$

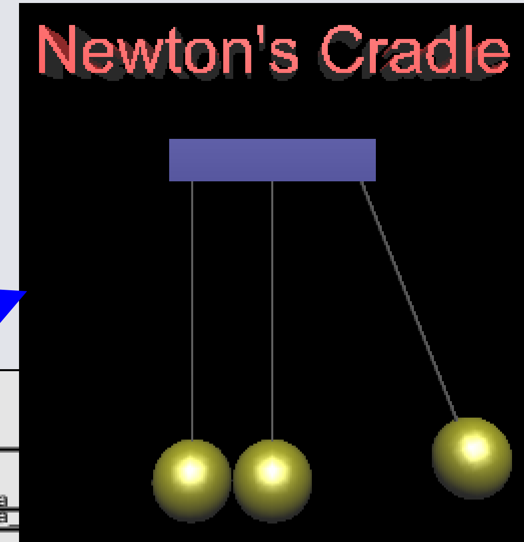
$S(P_3)=\{S_1,S_4,S_6\}$

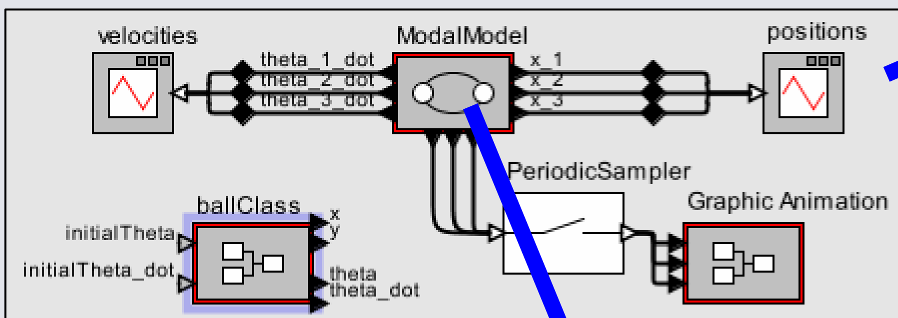
Metropolis II Infrastructure

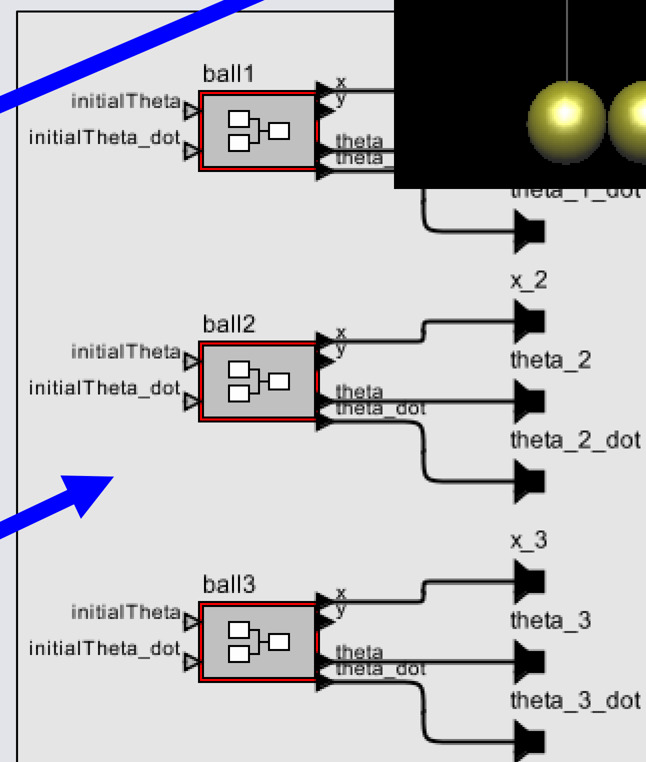


Ptolemy II and HyVisual

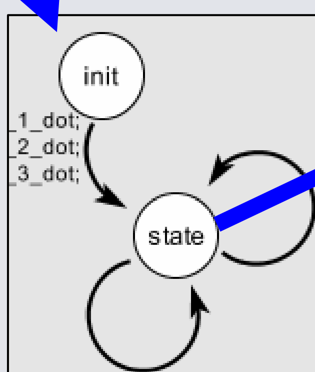
Unification of synchronous/reactive, discrete-event, and continuous-time semantics led to major redesign of hybrid systems modeler HyVisual.



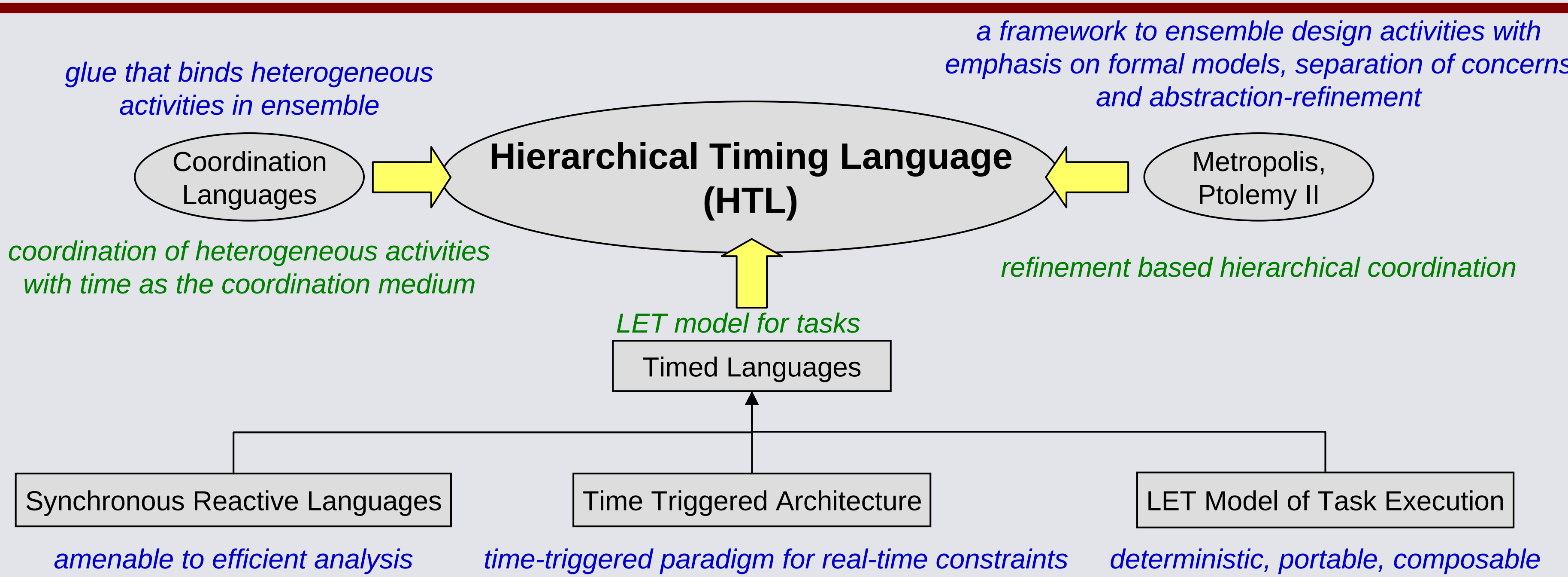




HyVisual is a specialization of the meta framework Ptolemy II.



Next Directions
HTL: Sources and Motivation



Coordination Languages

lack formal semantics	provide formal semantics for coordination

Metropolis/ Ptolemy II

general framework	provide a model for time triggered coordination with efficient verification scheme

Timed Languages

latency	reduce latency
restricted to one level	hierarchical refinement based structure
schedulability gets harder with expressiveness	efficient analysis while compact representation

HTL: Key Points and Relation to Ecosystem

- A coordination language (HTL) for hard real-time applications; HTL programs are extensible in two dimensions without changing timing behavior

- Time invariance under parallel composition (adding new program modules) is achieved by ensuring that different program modules communicate at specified instances of time

- Time invariance under vertical extension (refinement of individual tasks) is achieved by conservative scheduling of the top level