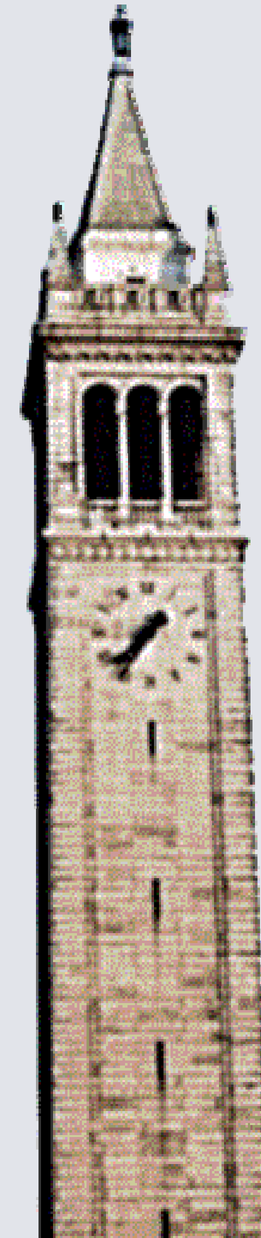
Advanced Tool Architectures

Edited and presented by
Edward A. Lee
UC Berkeley

Chess Review
November 21, 2005
Berkeley, CA

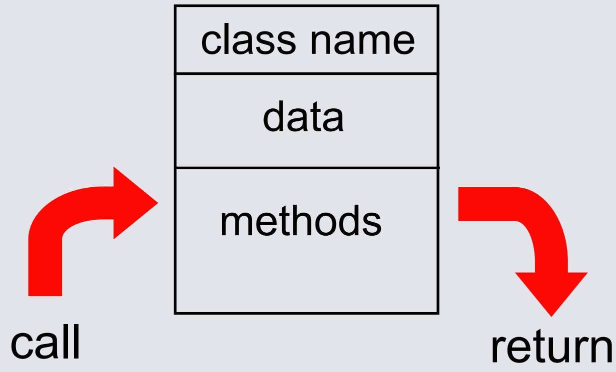


Thrust III: Advanced Tool Architectures

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

A Unifying Theme: Actor-Oriented Software Components

Object-oriented:

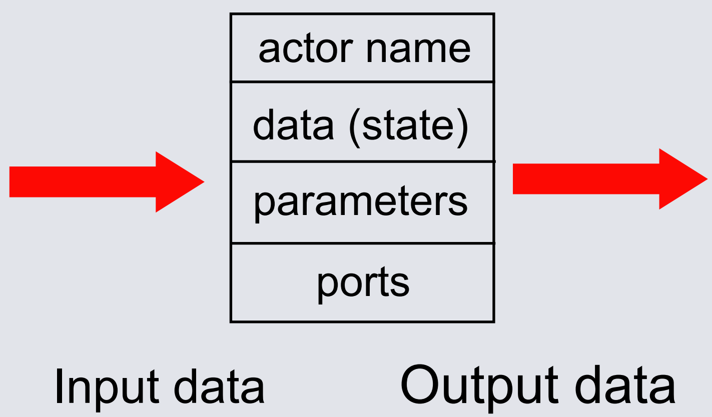


What flows through an object is sequential control

Things happen to objects

Actor oriented:

Actors make things happen



What flows through an object is streams of data

Examples of Actor-Oriented Platforms

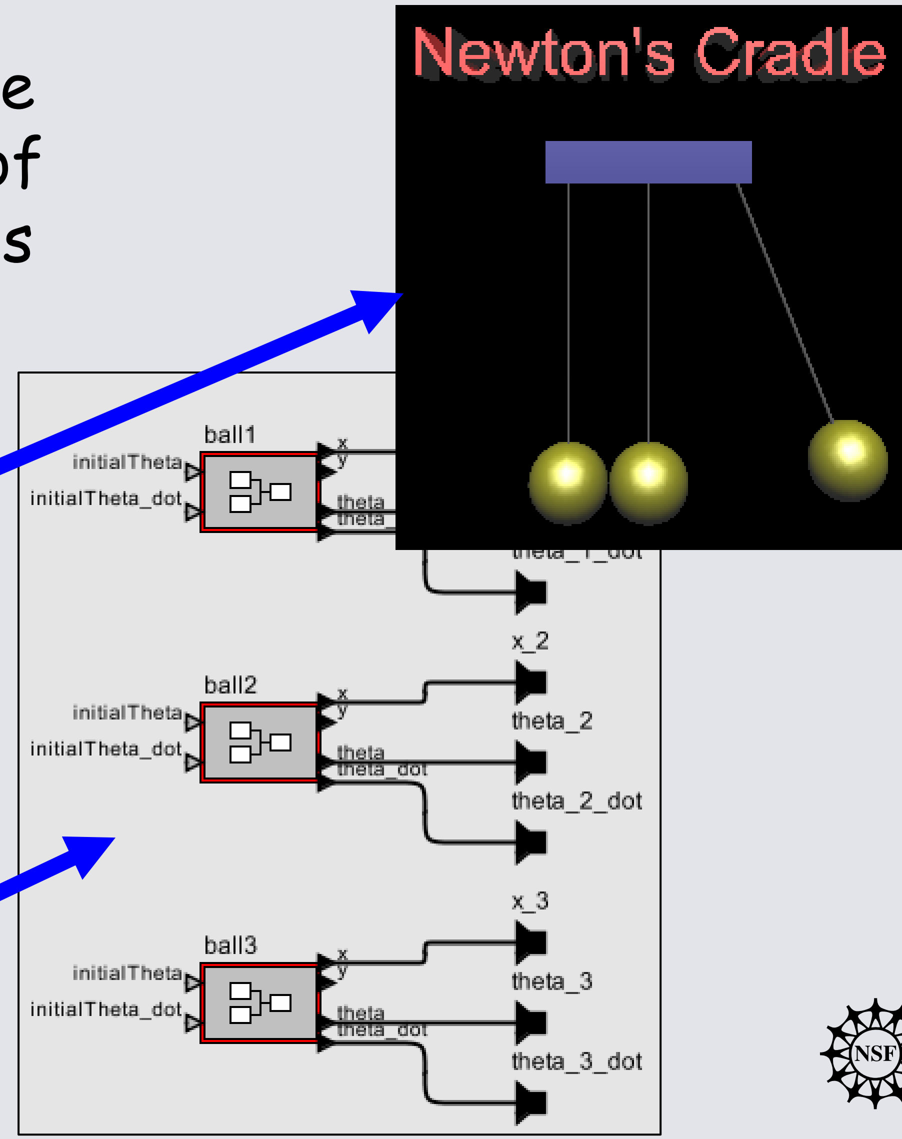
- Simulink (The MathWorks)
- LabVIEW (National Instruments)
- Modelica (Linkoping)
- OPNET (Opnet Technologies)
- Giotto and xGiotto (UC Berkeley)
- Polis & Metropolis (UC Berkeley)
- Gabriel, Ptolemy, and Ptolemy II (UC Berkeley)
- OCP, open control platform (Boeing)
- GME, actor-oriented meta-modeling (Vanderbilt)
- SPW, signal processing worksystem (Cadence)
- System studio (Synopsys)
- ROOM, real-time object-oriented modeling (Rational)
- Easy5 (Boeing)
- Port-based objects (U of Maryland)
- I/O automata (MIT)
- VHDL, Verilog, SystemC (Various)
- …

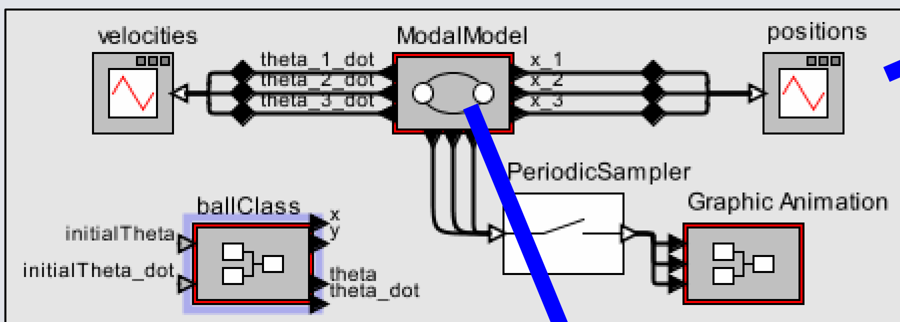
Major Advances in the Last Year

- Operational Semantics for Hybrid Systems
- Executable Stochastic Hybrid Systems
- Executing Beyond Zeno
- Composable Schedulability Analysis
- Improved Model Transformation Tools
- Interface Theories (for real time, causality, & refinement)
- Software releases:
  - GME
  - GReAT
  - HyVisual (Hybrid systems modeling)
  - Metropolis
  - Ptolemy II
  - UDM
  - Visualsense (Sensor network modeling)
  - Viptos (TinyOS + Visualsense)

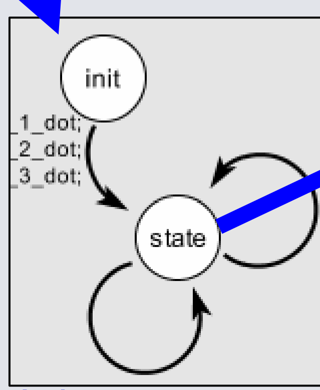
Operational Semantics for Hybrid Systems

Provides predictably executable models with rigorous handling of discontinuities and simultaneous events [Lee, Zheng]





HyVisual tool provides a modeling and simulation environment for hybrid systems.



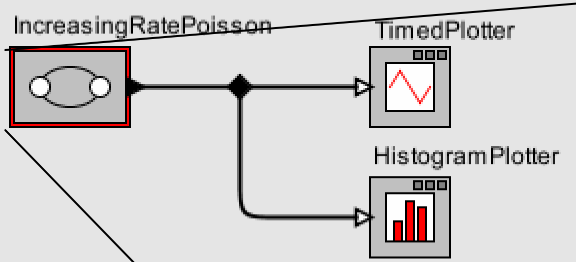
Executable Stochastic Hybrid Systems

Monte-Carlo models of stochastic hybrid systems are now supported by HyVisual.

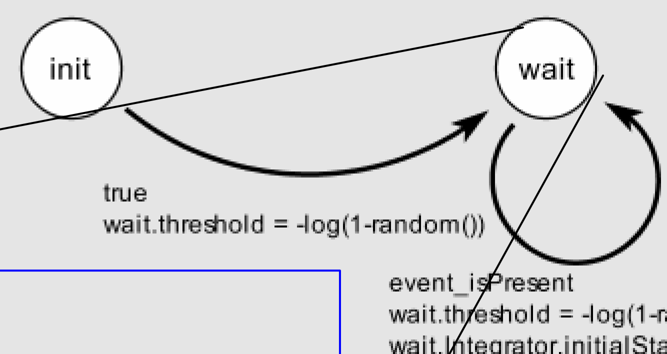
CT Director

● rateOfRateIncrease: 0.005

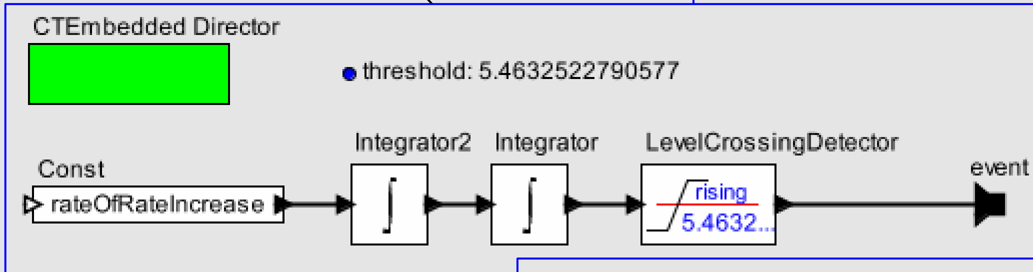
This model is an example of a stochastic hybrid system. It generates a Poisson process with a linearly increasing rate using the CT domain. The model plots the events vs. time and a histogram of the time between events. The technique here was suggested by John Lygeros.

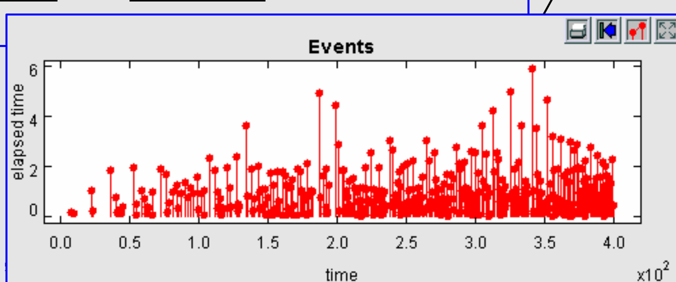


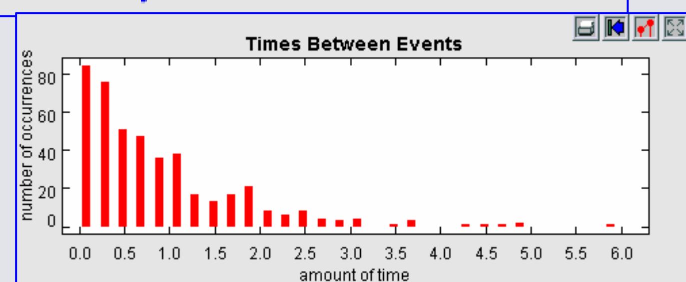
On each transition, generate a new random number with an exponential distribution. In the "wait" state, wait an amount of time that is the value of this random variable multiplied by the current (increasing) rate.



Note that on this transition, only one of the two integrators is reset, so the rate continues to increase linearly.



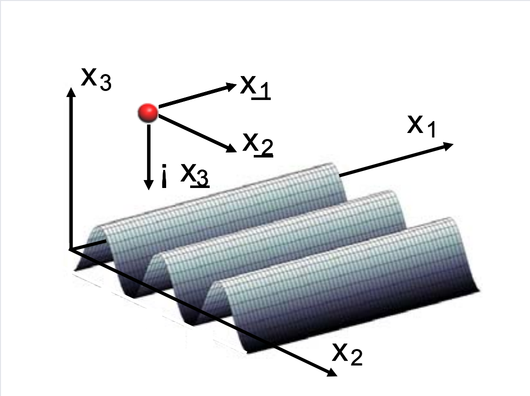


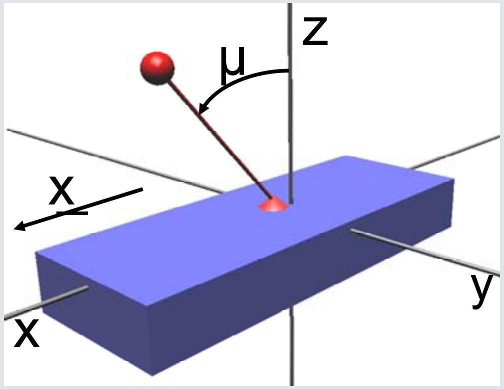


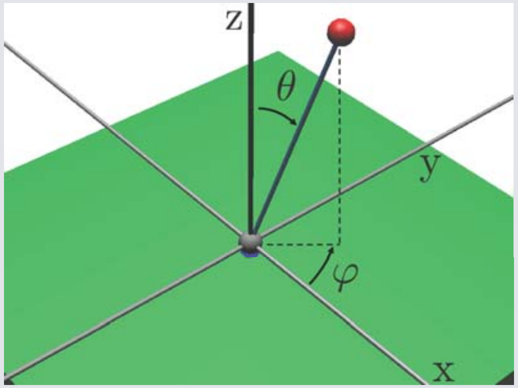
Executing Beyond Zeno

Provides systematic completion of models beyond Zeno point [Ames, Gregg, Lee, Sastry, Zheng].

Ball on Sinusoidal Surface	Pendulum on a Cart	Spherical Pendulum on the Ground







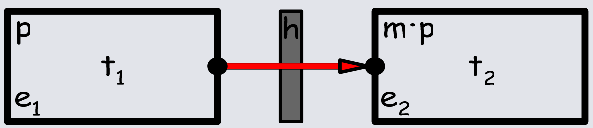
$h_B(x_1; x_2; x_3) = x_3 - \cos(x_2)$

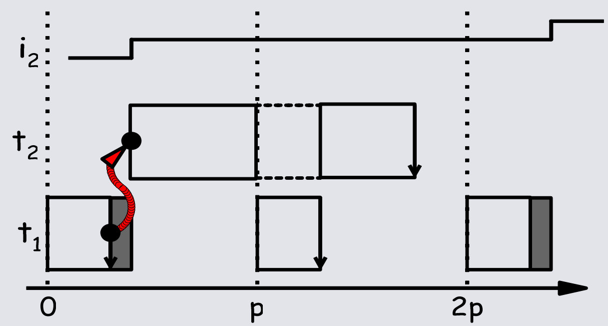
$h_C(\theta; x) = \cos(\theta)$

$h_P(\theta; \varphi) = \cos(\theta)$

Composable Real-Time Schedulability

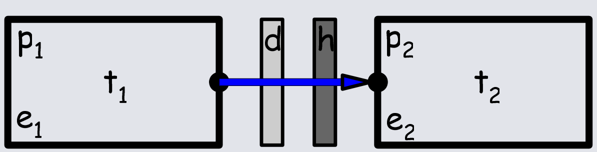
Simulink with Real-Time Workshop puts sample delays only on fast to slow inter-component connections, resulting in lower latency, but less compositionality.

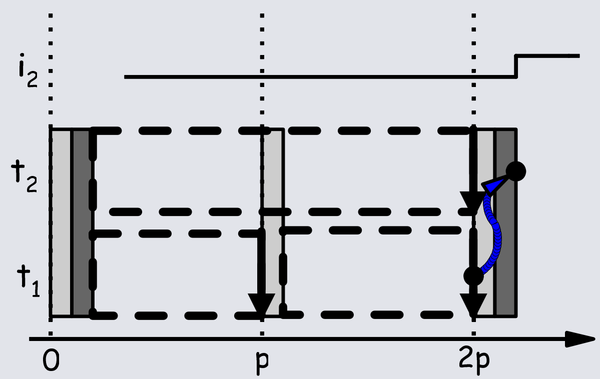




Simulink and Giotto both yield periodic real-time scheduling with deterministic results. Giotto yields higher latency, but better compositionality.

Giotto puts sample delays on every inter-component connection, resulting in higher latency, but better compositionality.



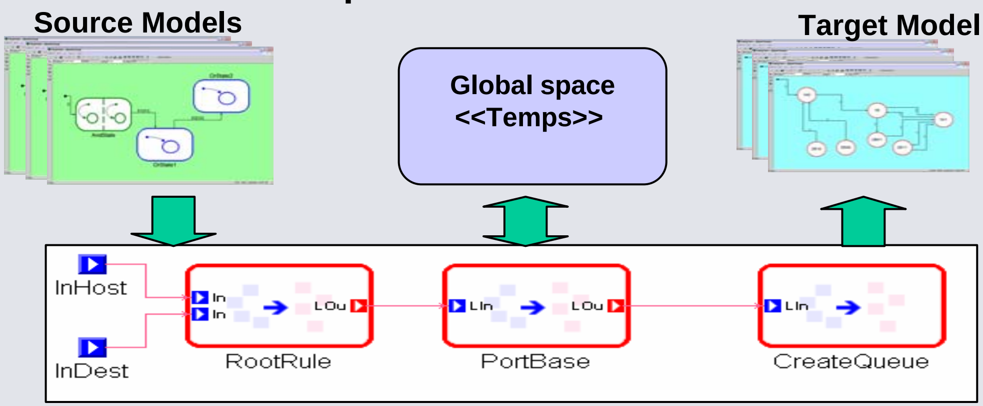


See poster presentation by Slobodan Matic.

Model Transformation Tools

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

Global "spaces" for Transformations

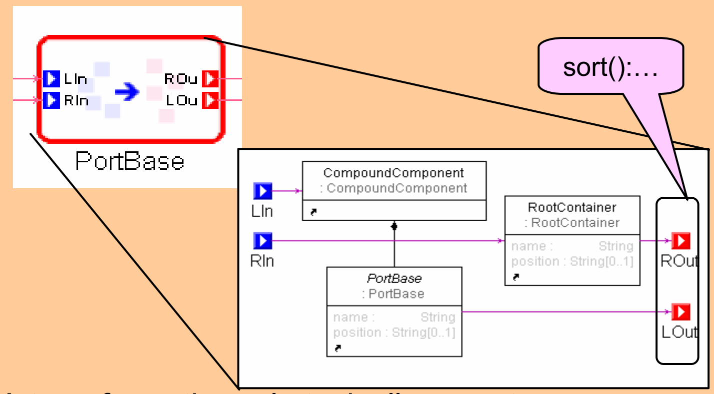


Global spaces hold intermediate results of the transformation
Consequence: The transformations are simplified.

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.
Consequence: 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

Interface Theories

Representing *Behavior* in Interfaces:

- Interaction semantics [Talcott, 1996]
- Tagged signal model [Lee, Sangiovanni-Vincentelli, 1997]
- Interface theories [de Alfaro, Henzinger, 2001]
  - E.g. Resource Interfaces [Chakrabarti, de Alfaro, Henzinger, 2003]
- Behavioral subtyping [Liskov, Wing, 1999]
- Behavioral type systems [Lee, Xiong, 2004]
- Agent Algebras [Passerone, Sangiovanni-Vincentelli, 2004]
- Abstract behavioral types [Arbab, 2005]

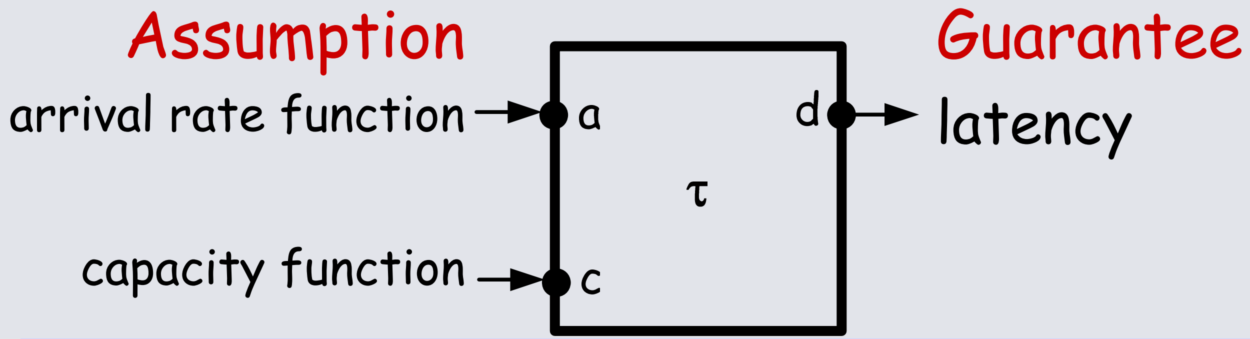
Major Current Efforts in Chess

Algebraic interface theories for:

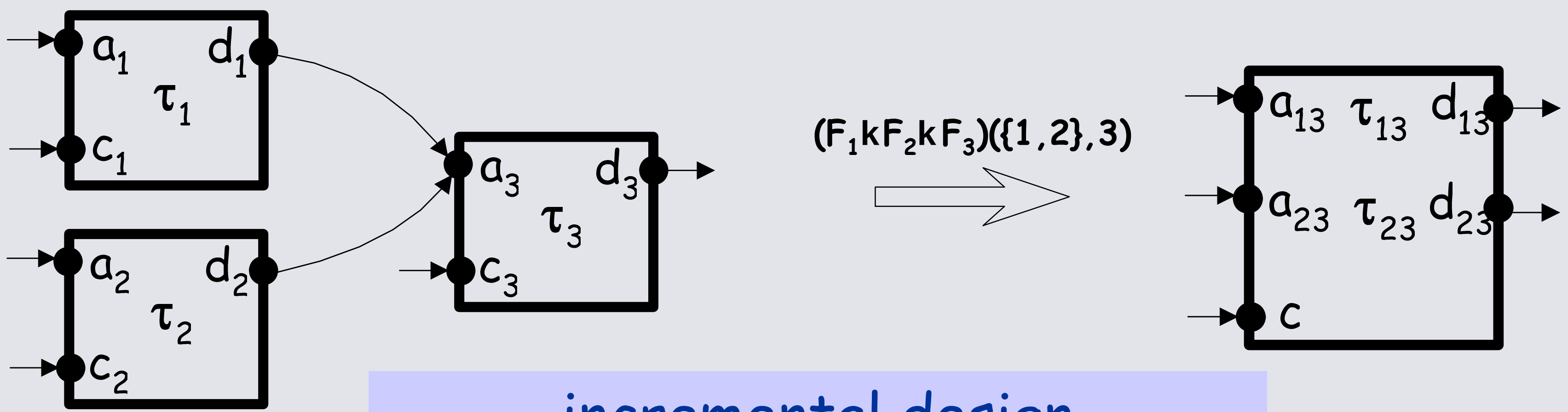
- Real-time
  - [Matic, Henzinger]
- Causality
  - [Lee, Zheng, Zhou]
- Refinement
  - [Passerone, Sangiovanni-Vincetelli]

Interface Algebra for Real Time

[Matic, Henzinger]



composition operation + refinement relation



incremental design

independent refinement

Interface Algebra for Refinement
[Passerone, Sangiovanni-Vincentelli]

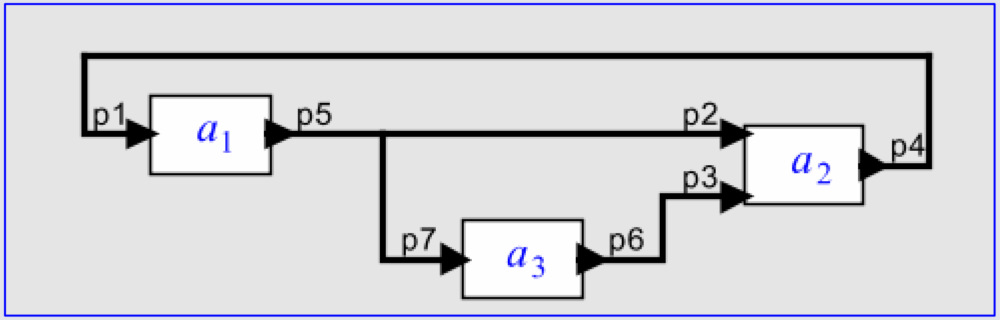
- Refinement in a model expressed as a relation ≤ of *agent substitutability*
- Yields a theory of compatibility
- Under certain necessary and sufficient conditions a model can be shown to have “mirrors”
  - A complement of an agent which is maximal relative to the compatibility relation
- Mirror operator used to link the notion of compatibility, refinement and composition
  - $p \le p'$ iff $p \,||\, \mathrm{mirror}(p') \in G$
  - p compat p' iff $p \le \mathrm{mirror}(p')$
- Mirrors used to solve the synthesis of the maximally compatible component in a context C under a specification S
  - $p \le \mathrm{mirror}(C \,||\, \mathrm{mirror}(S))$

Interface Algebra for Causality
[Lee, Zheng, Zhou]

With careful definition of the model of computation, actor-oriented models can have well-founded semantics. That is, any syntactically-correct model has a unique and well-defined meaning.

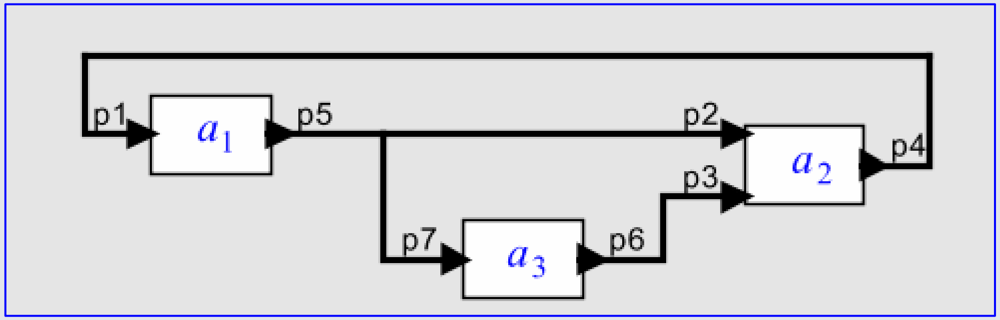
However, that meaning may not be useful. The model may suffer from:

- Causality loops
- Deadlock
- Algebraic loops



Collectively, these are all causality problems, and can be unified under a theory of *causality interfaces*.

Actor-Oriented Component Composition



- Cascade connections
- Parallel connections
- Feedback connections

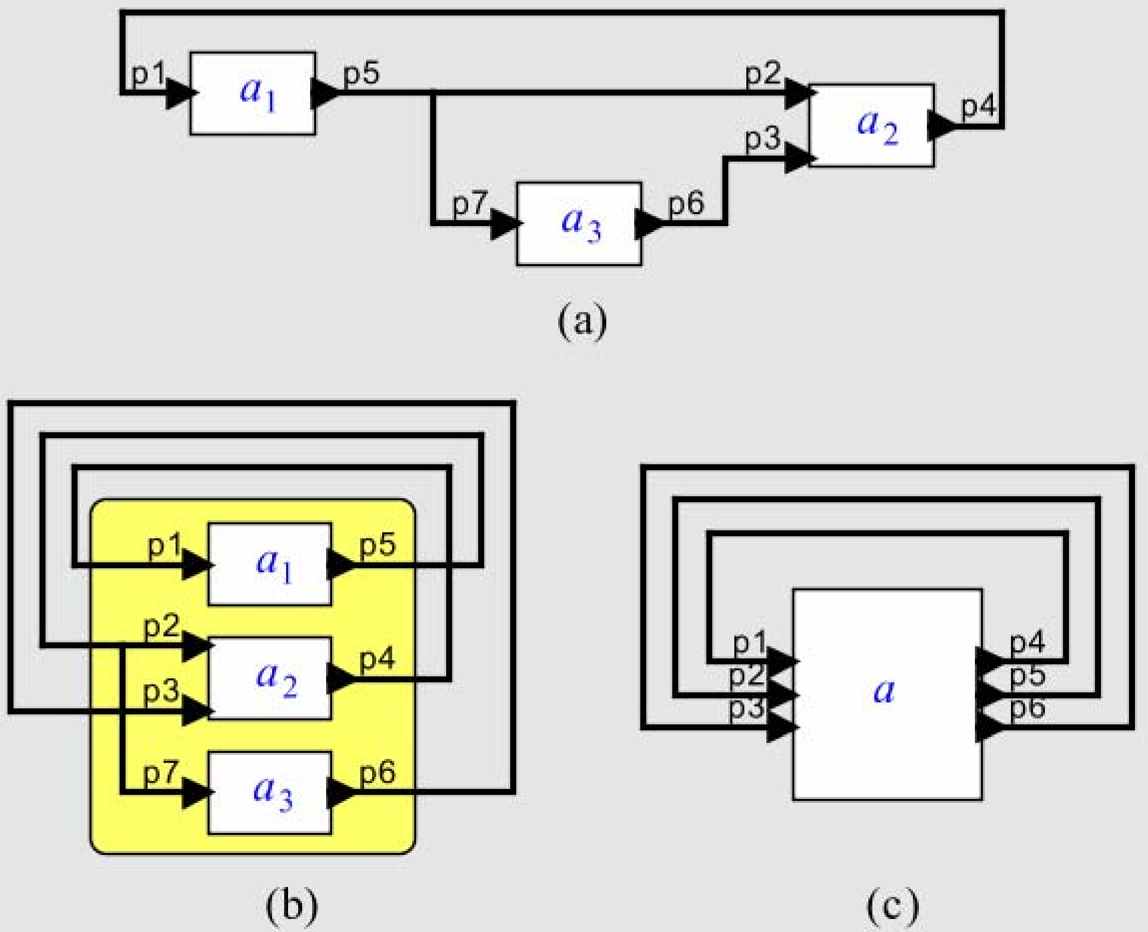
If actors are functions on signals, then the nontrivial part of this is feedback.

Some of the Possible Models of Computation:

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

All Actor Compositions are Feedback Compositions

Any composition of functional actors can be reduced to a feedback composition of a single functional actor.

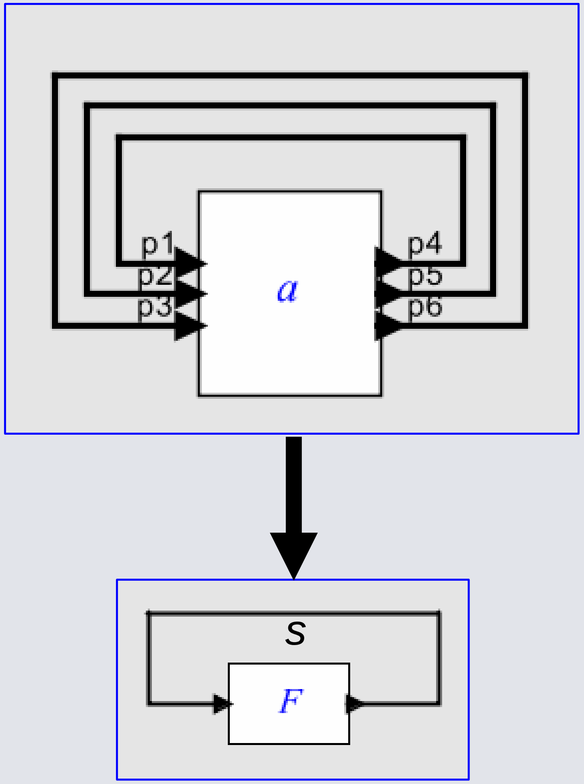


Feedback Form: Fixed-Point Semantics

For functional actors, semantics is a fixed point.

Unique least fixed point exists if actors are monotonic functions on a CPO (process networks, dataflow, synchronous/reactive)

Unique fixed point exists if actors are contraction maps on a metric space (discrete events).



Signal *s* satisfies

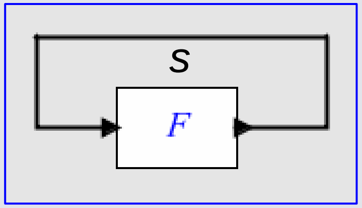
$$F(s) = s$$

This is called a fixed-point of the function *F*.

But Existence of a Fixed Point Doesn't Ensure a Useful Behavior!

To get a useful behavior we need:

1. That the fixed point *s* can be found constructively (constructive semantics)

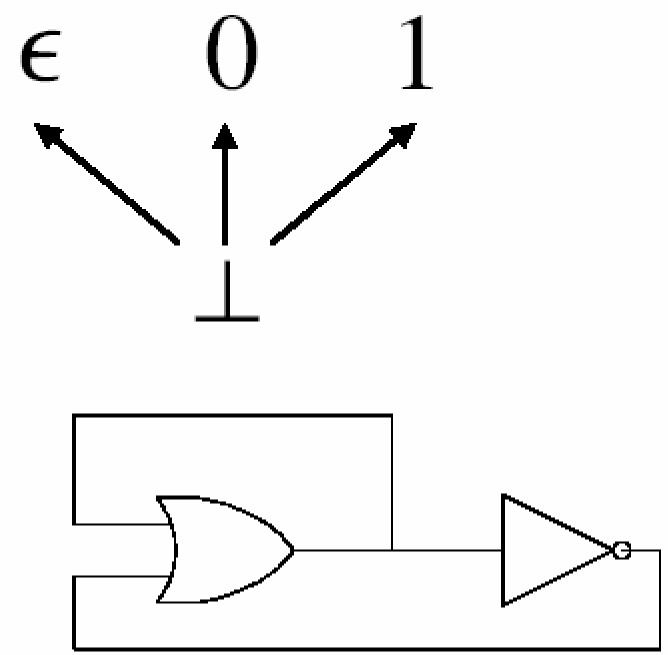


2. That the fixed point *s* not define signals to be unknown (causality loops)

3. That the fixed point *s* not define empty signals, or, sometimes, finite signals (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.



$(\bot, \bot)$ 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.

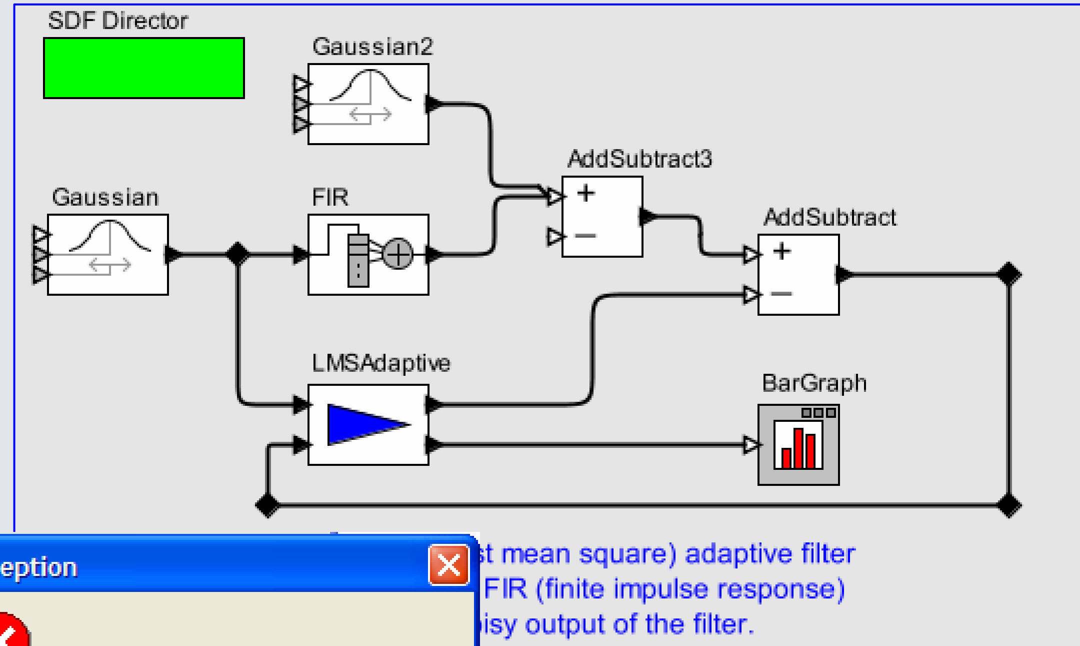
⊥ is the only fixed point solution.

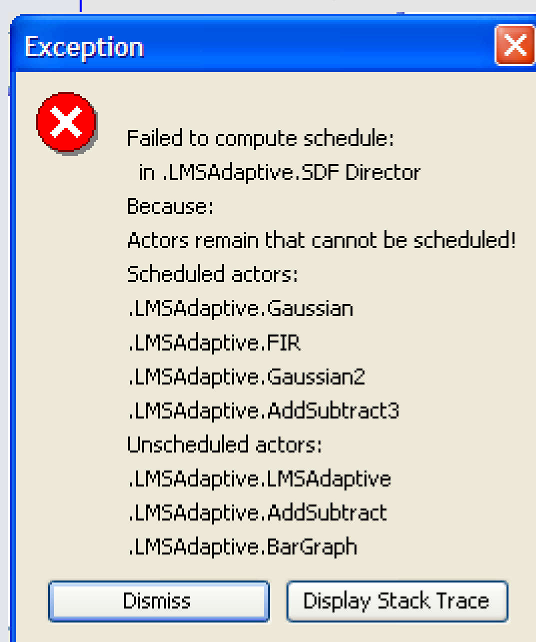
⊥, 0, and 1 are all fixed point solutions.

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.





Causality Interfaces

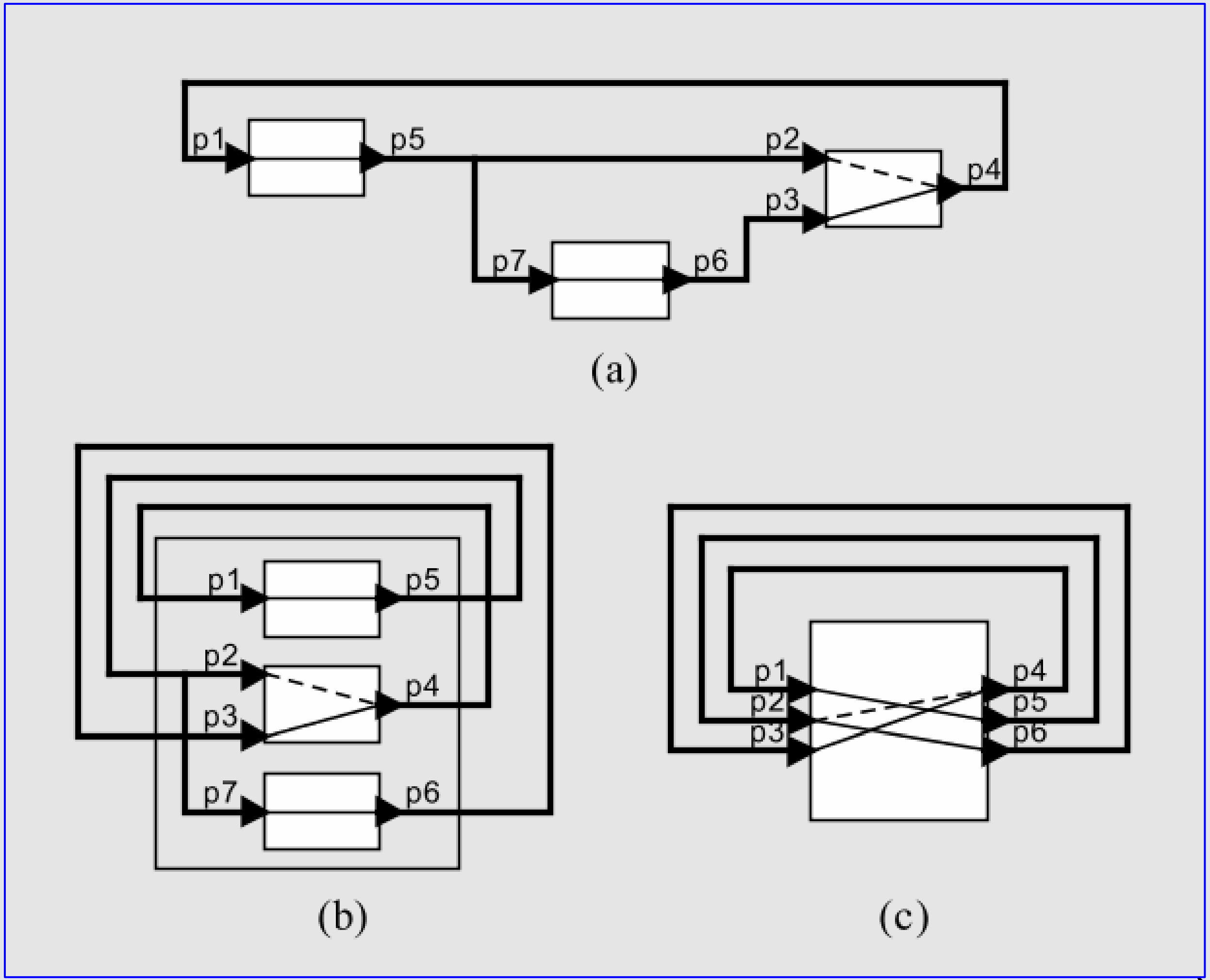
Causality interfaces provide the analytical toolkit that identifies these defects when they can be identified (i.e. when they are decidable):

- Causality loops in synchronous programs;
- Delay-free cycles in discrete-event models;
- Deadlock in dataflow models.

The same algebraic structure works for all of these.

Causality Interfaces

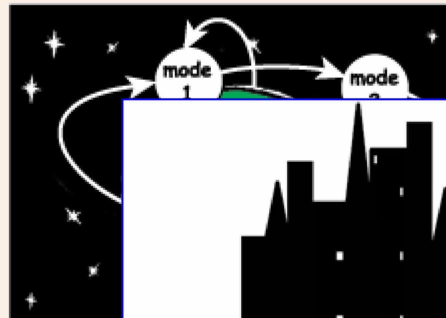
An algebra of interfaces provides operators for cascade and parallel composition and necessary and sufficient conditions for causality loops, zero-delay loops, and deadlock.



See poster presentation by Rachel Zhou

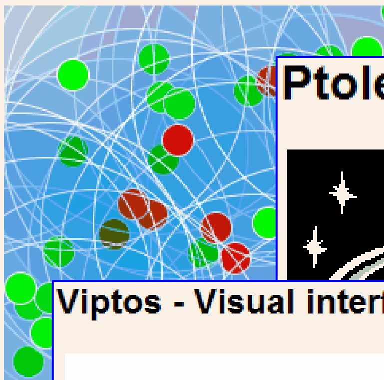
Software Releases

HyVisual - Hybrid System Visual Modeler



VisualSense

Visual editor and simulator for wireless sensor network systems



Ptolemy II

Viptos - Visual interface between Ptolemy and TinyOS

Metropolis

Metropolis: Design Environment for Heterogeneous Systems

Universal Data Model (UDM)

The Generic Modeling Environment

GME 5

The Graph Rewrite And Transformation (GReAT) tool suite

The Hyper toolbox (in development)

- Making sense of Multiple Tools
- Consider Interchange Format Philosophy:
  - For all models which *could be* built in $Tool_1$ or $Tool_2$ (i.e., as defined by A_1) there must exist a translator to/from an Interchange Format
- Alternative philosophy:
  - For a model, *m*, built in $Tool_1$ or $Tool_2$, this model may be translated to the other tool *if* the semantics used by *m* are an intersecting subset of the semantics $S_1 \cap S_2$.

$Tool_1 = \langle C_1, A_1, S_1, M_{s1}, M_{c1} \rangle$

C = Concrete Syntax, A = Abstract Syntax, S = Semantics
M_s = Semantic Mapping, M_c = Concrete Syntax Mapping

The Hyper toolbox (in development)

- Examine semantics used by a model to determine compatibility
- This provides several potential uses
  - Produce $\text{Tool}_{1\cap 2}$ after user request for models compatible across Tool_1, Tool_2
  - Check to see if model m_3, produced in $\text{Tool}_{1\cap 3}$ is compatible with Tool_2
  - Produce $\text{Tool}_{\text{simulate}\cap\text{verify}}$ when capability is more important than specific semantics
- Implementation strategy
  - Strong typing, metamodeling of type structures
  - Previous Chess work in operational semantics and Interchange Formats

Major Ongoing Efforts

- Abstract Semantics
- Interface Theories
- Scalability in Actor-Oriented Design
- Model Transformations and Code Generation
- Hybrid Systems Tool Interaction (Hyper)
- Software Tools
  - GReAT
  - HyVisual
  - Visualsense
  - Viptos
- Meta frameworks
  - GME
  - Metropolis
  - Ptolemy II
  - UDM