Advanced Tool Architectures

Edited and presented by
Edward A. Lee
UC Berkeley
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:**

- Class name
- Data
- Methods

What flows through an object is sequential control. Things happen to objects.

**Actor oriented:**

- Actor name
- Data (state)
- Parameters
- Ports

What flows through an object is streams of data. Actors make things happen.

"Advanced Tool Architectures", E.A. Lee
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.

"Advanced Tool Architectures", E.A. Lee
Executable Stochastic Hybrid Systems

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.

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

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

\[ h_B(x_1; x_2; x_3) = x_3 i \cos(x_2) \]

\[ h_C(\mu; x) = \cos(\mu) \]

\[ h_P(\mu;') = \cos(\mu) \]
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.
Global “spaces” for Transformations

Source Models

Global space <<Temps>>

Target Model

InHost

RootRule

Local

InDest

PortBase

CreateQueue

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

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

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.
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]

**Assumption**
arrival rate function → \( a \)

capacity function → \( c \)

**Guarantee**
l latency

**composition operation + refinement relation**

\((F_1 F_2 F_3)((1,2), 3)\)

**incremental design**
independent refinement

"Advanced Tool Architectures", E.A. Lee

Chess Review, Nov. 21, 2005
• Refinement in a model expressed as a relation $\leq$ 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
  - $p \leq p'$ iff $p \parallel \text{mirror}(p') \in G$
  - $p \text{ compat } p'$ iff $p \leq \text{mirror}(p')$

• Mirrors used to solve the synthesis of the maximally compatible component in a context $C$ under a specification $S$
  - $p \leq \text{mirror}(C \parallel \text{mirror}(S))$
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.
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 $F$. 

"Advanced Tool Architectures", E.A. Lee
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.

(⊥, ⊥) 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.

\[\perp\] is the only fixed point solution. \[\perp, 0, \text{and } 1\] 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

VisualSense
Visual editor and simulator for wireless sensor network systems

Ptolemy II

Viptos - Visual interface between Ptolemy and TinyOS

HyVisual - Hybrid System Visual Modeler

Metropolis: Design Environment for Heterogeneous Systems

The Generic Modeling Environment

GME 5

Universal Data Model (UDM)

The Graph Rewrite And Transformation (GReAT) tool suite

"Advanced Tool Architectures", E.A. Lee
The Hyper toolbox (in development)

- Making sense of Multiple Tools
- Consider Interchange Format Philosophy:
  - For all models which could be built in Tool₁ or Tool₂ (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 Tool₁ or Tool₂, this model may be translated to the other tool if the semantics used by \( m \) are an intersecting subset of the semantics \( S_1 \cap S_2 \).

\[
\text{Tool}_1 = \langle C_1, A_1, S_1, M_{s1}, M_{c1} \rangle
\]

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 $\text{Tool}_1 \cap \text{Tool}_2$ after user request for models compatible across $\text{Tool}_1$, $\text{Tool}_2$
  - Check to see if model $m_3$, produced in $\text{Tool}_1 \cap \text{Tool}_3$ is compatible with $\text{Tool}_2$
  - 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