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
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.

\[ F : S_1 \rightarrow S_2 \]
\[ s_1 \in S_1 \]
\[ f : S_1 \times \Sigma \rightarrow S_2 \]
\[ g : S_1 \times \Sigma \rightarrow \Sigma \]

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

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

"Advanced Tool Architectures," Edward A. Lee
Major Concepts 3: Experimental Tools & Frameworks

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
What We Have Learned

Hybrid and embedded software systems demand a different approach to computation.
Instead of a Program Specifying...

\[ f : \{0,1\}^* \rightarrow \{0,1\}^* \]
... A Program Should Specify

\[ f : [T \rightarrow \{0,1\}^\ast]^P \rightarrow [T \rightarrow \{0,1\}^\ast]^P \]

"actor"  "signal"  "signal"

...where \( T \) is a set representing time, precedence ordering, causality, synchronization, etc.
The Catch…

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

- 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:

<table>
<thead>
<tr>
<th>class name</th>
<th>data</th>
<th>methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What flows through an object is sequential control

Things happen to objects

The alternative: "Actor oriented:"

<table>
<thead>
<tr>
<th>actor name</th>
<th>data (state)</th>
<th>parameters</th>
<th>ports</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Actors make things happen

What flows through an object is evolving data

Input data       Output data

"Advanced Tool Architectures," Edward A. Lee
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

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

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
- ...

"Advanced Tool Architectures," Edward A. Lee
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
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.

\[ \perp \] 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.
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.
Global “spaces” for Transformations

Source Models

Global space <<Temps>>

Target Model

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

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.
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

\[
\begin{align*}
S(P_1) &= \{S_1, S_2, S_3\} \\
S(P_2) &= \{S_3, S_4, S_5\} \\
S(P_3) &= \{S_1, S_4, S_6\}
\end{align*}
\]
Metropolis II Infrastructure

Object
1. Input port
2. Output port
3. View port
4. Assumption
5. Guarantee
6. Object Type

Object Types
- MetaModel
- C/C++
- VHDL/Verilog
- Pre-compiled

Object
- Compiler
- Elaborator
- Simulator
- Formal Verifier
- Synthesizer

Syntax Information
Semantics Information
Structure Information

"Advanced Tool Architectures," Edward A. Lee
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
  - glue that binds heterogeneous activities in ensemble
  - coordination of heterogeneous activities with time as the coordination medium

- Hierarchical Timing Language (HTL)
  - LET model for tasks

- Metropolis, Ptolemy II
  - refinement based hierarchical coordination

- Synchronous Reactive Languages
  - amenable to efficient analysis

- Time Triggered Architecture
  - time-triggered paradigm for real-time constraints

- LET Model of Task Execution
  - deterministic, portable, composable

---

**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
- restricted to one level
- schedulability gets harder with expressiveness
- reduce latency
- hierarchical refinement based structure
- 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.