Correctly Composing Components: Ontologies and Modal Behaviors

Edward A. Lee
Robert S. Pepper Distinguished Professor
UC Berkeley

With: Ben Lickly, Man-Kit Leung, Thomas Mandl, Elizabeth Latronico (Bosch), Charles Shelton (Bosch), Stavros Tripakis

MURI Review

December 2, 2009
Berkeley, CA
A Taxonomy of Modeling Issues

Abstract Syntax (static structure)
[software architecture, metamodeling, higher-order components, ...]

Static Semantics (type systems)
[type inference/checking, ontologies, behavioral types, ...]

Dynamic Semantics (models of computation)
[automata, hybrid systems, model models, tagged signal model, Kahn networks, quantitative system theory, ...]

Ontologies

Modal Models
Reporting Progress in Two Dimensions of Model Engineering

“Model engineering” is the “software engineering” of models. How to build, maintain, and analyze large models.

I will talk about two specific accomplishments:

- **Model ontologies (static semantics)**
  - Check for compatible static semantics in pieces of models
  - Using semantic property annotations and inference
  - Based on sound foundations (type theories)
  - Scalable to large models

- **Modal models (dynamic semantics, a form of multimodeling)**
  - Components of a model with distinct modes of operation
  - Switching between modes is governed by a state machine
  - State machines composable with many concurrency models
  - Hybrid systems are a special case
Static Semantics

*First*: capture domain-specific semantic information

- Components in a model (e.g. parameters, ports) can have properties drawn from a lattice.
- Components in a model (e.g. actors) can impose constraints on property relationships.
- The type system infrastructure can infer properties and detect errors.

Example of a simple domain-specific semantic lattice (an ontology) for vehicle motion models.

Another example of an ontology for model optimization.
Static Semantics

Second: Define constraints across components

\[
\int \text{acceleration}(t) \, dt = \text{speed}(t)
\]

\[
\int \text{speed}(t) \, dt = \text{position}(t)
\]
Static Semantics

Third: annotate the model

DimensionSystemSolver

Double click to Resolve Properties

ConstNonconstSolver

Double click to Resolve Properties

PropertyRemover

Double click to Remove Properties

Const

1

Integrator

Display

DimensionSystemSolver::Constraint: Const.output == Acceleration
Static Semantics

**Fourth: Run the solver.**
Static Semantics
Fifth: Resolve inconsistencies exposed by the solver.
Applying this ontology to a model: Cooperative control system

This model shows a simple cooperative control system illustrating model-integrated control strategies. A leading vehicle produces information that is observed with possible flaws by a following vehicle. If the following vehicle detects flaws, it uses a conservative strategy. Otherwise, it matches the speed of the leaving vehicle.

For simplicity, this model handles only one-dimensional motion.
Applying this ontology to a model: Cooperative control system
Solver infers ontology information throughout the model and checks for consistent usage.

Vehicle simulator. This model takes as input a desired speed and implements a simple proportional controller with the specified loop gain to achieve that speed. It outputs the acceleration, speed, and position of the vehicle.
Background for Model Ontologies: Hindley-Milner Type Theories

- A *lattice* is a partially ordered set (poset) where every subset has a least upper bound (LUB) and a greatest lower bound (GLB).

- Modern type systems (including the Ptolemy II type system, created by Yuhong Xiong) are based on efficient algorithms for solving inequality constraints on lattices.
Relational Constraint Problem (RCP)

\[ RCP : (P, C) \]

\( P \) is a partially ordered set, \( C \) is a set constraints of the form:

\[ r(p_x, p_y, \ldots) \]

where \( r \) is a relation (e.g. \( =, \leq \)).

A solution is a satisfying assignment to property variables \( p_x, p_y, \ldots \).
Definite Monotone Function Problem (DMFP)

Monotonic Function
A function $f$ for which
$x_1 \leq x_2 \implies f(x_1) \leq f(x_2)$

Special case of RCP

$DMFP : (P, C_F)$

$P$ is a lattice, $C_F$ is a set of definite inequalities:

$f(p_y, p_z, \ldots) \leq p_x$

where $f$ is a monotonic function.

Here, there is a unique least fixed point (LFP) solution, efficiently solved by an algorithm given by Rehof and Mogensen (1996).
Problem Statement

Given:

Lattice: \( P \) \hspace{1cm} (1)

Constants & Variables: \( p_1, p_2, \ldots \in P \) \hspace{1cm} (2)

Constraints of the form: \( f(p_1, p_2, \ldots) \leq p_n \) (\( f \) monotonic) \hspace{1cm} (3)

Is there a satisfying assignment to variables?

This problem has a linear time algorithm!
(Rehof and Mogensen, 1996)
How to Make this Usable in Practice?

A problem is that, in general, the number of constraints is proportional to the size of the model.

To mitigate these, organize constraints as:

- **Default Constraints**
  - Set globally by the property solver
  - (actors, connections, etc.)

- **Actor-specific Constraints**
  - Use an adapter pattern for actors

- **Instance-specific Constraints**
  - Specified through model annotations
Example of Actor Constraints for the Dimensions Lattice

$$f_I(p_y) = \begin{cases} 
\text{Undef.} & \text{if } p_y = \text{Undef.} \\
\text{Speed} & \text{if } p_y = \text{Pos.} \\
\text{Accel.} & \text{if } p_y = \text{Speed} \\
\text{Unitless} & \text{if } p_y = \text{Time} \\
\text{Error} & \text{otherwise} 
\end{cases}$$

$$f_O(p_x) = \begin{cases} 
\text{Undef.} & \text{if } p_x = \text{Undef.} \\
\text{Pos.} & \text{if } p_x = \text{Speed} \\
\text{Time} & \text{if } p_x = \text{Accel.} \\
\text{Error} & \text{otherwise} 
\end{cases}$$
Another Lattice

This example illustrates that an ontology can be used to determine in which parts of a model signals vary dynamically.
Another Lattice

This example illustrates that an ontology can be used to determine in which parts of a model signals vary dynamically.
Reporting Progress in Two Dimensions of Model Engineering

“Model engineering” is the “software engineering” of models. How to build, maintain, and analyze large models.

I will talk about two specific accomplishments:

- **Model ontologies (static semantics)**
  - Check for compatible static semantics in pieces of models
  - Using semantic property annotations and inference
  - Based on sound foundations (type theories)
  - Scalable to large models

- **Modal models (dynamic semantics, a form of multimodeling)**
  - Components of a model with distinct modes of operation
  - Switching between modes is governed by a state machine
  - State machines composable with many concurrency models
  - Hybrid systems are a special case
What is the meaning of modal behavior?

Modal models are formal representations of dynamically changing behaviors, where the changes are modeled by a state machine. They can be used to construct fault models and models of adaptive systems that react to faults.

I will describe a semantics of modal models embracing concurrent and timed models.
Motivating Example: Hybrid System

Finite State Machine

Guard: \( \text{abs(Force)} > \text{Stickiness} \)
Set: \( \text{Separate.p1} = P1; \)
\( \text{Separate.p2} = P2; \)
\( \text{Separate.v2} = V1 \)

Guard: \( \text{touched.isPresent} \&\& (V1-V2) > 0.0 \)
Set: \( \text{Together.p} = P1; \)
\( \text{Together.v} = (V1+V2)/2.0; \)
\( \text{Together.stickiness} = 10.0 \)

Concurrent Model

Expression
\[ 1.0 \times 1.0 - 1.0 \times P1 \]
\[ \frac{\partial}{\partial t} \]
P1
V1 integrator
V1
P1 integrator

Expression
\[ 2.0 \times 2.0 - 2.0 \times P2 \]
\[ \frac{\partial}{\partial t} \]
P2
V2 integrator
\( \text{P2} \)

AddSubtract
ZeroCrossingDetector
both
0.0
touched

Expression
\[ \text{touched.isPresent} \&\& (V1-V2) > 0.0 \]
\[ \frac{\partial}{\partial t} \]

Expression
\[ 1.0 \times 1.0 - 2.0 \times 2.0 - (1.0-2.0) \times P1 \]
\[ \frac{\partial}{\partial t} \]

V1 and V2 are velocities, and P1 and P2 are positions of the two masses.

Lee, Berkeley 22
Generalizing Beyond Hybrid Systems

Hybrid systems define modal behavior in continuous-time dynamics. We are generalizing this to give a modal semantics to discrete time, discrete-event, and untimed models.

Cooperative control system example includes two timed modal models. E.g.
Meta Model for FSMs in Ptolemy II

- Initial state indicated in bold
- Guards are expressions that can reference inputs and variables
- Output values can be functions of inputs and variables
- Transition can update variable values ("set" actions)
- Final state terminates execution of the actor

Lee, Berkeley 24
Extended State Machines
Reference and manipulate variables on guards and transitions.

Extended state machines can operate on variables in the model, like “count” in this example.

“Set” actions are distinct from “output” actions. We will see why.
An actor’s behavior may be defined by an arbitrarily deep nesting of FSMs and refinements.

Director determines semantics of the submodel.
Ptolemy II Enables Hierarchical Mixtures of MoCs

This model has two simple synchronous/reactive (SR) models as mode refinements and models their timed environment using a discrete-event (DE) director.
Compare with Statecharts AND states

Here, two FSMs are composed under a synchronous/reactive director, resulting in Statecharts-like AND states.

Using a synchronous/reactive (SR) director yields Statechart-like semantics for concurrent state machines.
Operational Semantics: Firing

An actor’s behavior may be defined by an arbitrarily deep nesting of FSMs and refinements.
Operational Semantics: Postfiring

State changes are committed only in postfire, enabling fixed point iteration by using only firing.
Directors Benefiting from Fire/Postfire Separation (which we call the *Actor Abstract Semantics*)

- **Synchronous/Reactive (SR)**
  - Execution at each tick is defined by a least fixed point of monotonic functions on a finite lattice, where bottom represents “unknown” (getting a constructive semantics)

- **Discrete Event (DE)**
  - Extends SR by defining a “time between ticks” and providing a mechanism for actors to control this. Time between ticks can be zero (“superdense time”).

- **Continuous**
  - Extends DE with a “solver” that chooses time between ticks to accurately estimate ODE solutions, and fires all actors on every tick.

[Lee & Zheng, EMSOFT 07]
Handling Time in Modal Models

After trying several variants on the semantics of modal time, we settled on this:

A mode refinement has a *local* notion of time. When the mode refinement is inactive, local time does not advance. Local time has a monotonically increasing gap relative to global time.
Modal Time
Example

DiscreteClock generates regularly spaced events that trigger mode transitions.

These transitions are “history” transitions, so mode refinements preserve state while suspended.

Produce regularly spaced events in this mode.

Produce irregularly spaced events in this mode.
Modal Time Example

Mode transitions triggered at times 0, 2.5, 5, 7.5, etc.

Events with value 1 produced at (local times) 0, 1, 2, 3, etc.

First regular event generated at (global time) 0, then transition is immediately taken. First irregular event generated at (global time) 0, one tick later (in superdense time).

Local time 1 corresponds to global time 3.5 here.
Variant using Preemptive Transition

First regular event is not produced until global time 2.5 (local time 0).
Time Delays in Modal Models

Triggers transitions at (global times) 0, 1, 2, 3, ... 

First output is the second input to the modal model, which goes through the noDelay refinement.

Second output is the first input to the modal model, which goes through the delay refinement, which is inactive from time 0 to 1.
Variants for the Semantics of Modal Time that we Tried or Considered, but that Failed

- Mode refinement executes while “inactive” but inputs are not provided and outputs are not observed.
- Time advances while mode is inactive, and mode refinement is responsible for “catching up.”
- Mode refinement is “notified” when it has requested time increments that are not met because it is inactive.
- When a mode refinement is re-activated, it resumes from its first missed event.

All of these led to some very strange models…

Final solution: Local time does not advance while a mode is inactive. Growing gap between local time and global time.
Conclusion

Abstract Syntax
(static structure)
[software architecture,
metamodeling,
higher-order components, …]

Static Semantics
(type systems)
[type inference/checking,
ontologies,
behavioral types, …]

Dynamic Semantics
(models of computation)
[automata, hybrid systems,
model models,
tagged signal model,
Kahn networks,
quantitative system theory, …]

Ptolemy II Property System:
• User-defined ontologies
• A few model annotations
• Inference engine
• Consistency checker
• Scalable to large models

Ptolemy II Modal models:
• Modal behavior as FSMs
• Arbitrarily deep hierarchy
• Heterogeneous hierarchy
• A semantics of time

http://chess.eecs.berkeley.edu/pubs/611.html
http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-151.html
Acknowledgments:
The Ptolemy Pteam

Plus (not shown):
• Elizabeth Latronico (Bosch)
• Charles Shelton (Bosch)
• Stavros Tripakis (UCB)
More Variants of Modal Models Supported in Ptolemy II

- Transition may be a reset transition
  - Destination refinement is initialized

- Multiple states can share a refinement
  - Facilitates sharing internal actor state across modes

- A state may have multiple refinements
  - Executed in sequence (providing imperative semantics)
Still More Variants

- Transition may have a refinement
  - Refinement is fired when transition is chosen
  - Postfired when transition is committed
  - Time is that of the environment
And Still More Variants

Transition may be a “default transition”
- Taken if no non-default transition is taken
- Compare with priorities in SyncCharts

FSMs may be nondeterminate
- Can mark transitions to permit nondeterminism

This example is a hierarchical FSM showing a thermostat with a nondeterminate fault mode.