Ensuring Correct Composition of Components using Lattice-based Ontologies

Ben Lickly\textsuperscript{1}  Man-Kit Leung\textsuperscript{1}  Thomas Mandl\textsuperscript{2}  Edward A. Lee\textsuperscript{1}  Elizabeth Latronico\textsuperscript{2}  Charles Shelton\textsuperscript{2}  Stavros Tripakis\textsuperscript{1}

\textsuperscript{1}University of California, Berkeley

\textsuperscript{2}Bosch Research, LLC

WFCD - Foundations and Applications of Component-based Design in Conjunction with ES Week
Grenoble, France, Oct. 11, 2009
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, synchronous languages, BIP, Broy’s components, tagged signal model, Kahn networks, Henzinger’s quantitative system theory, ...]
Abstract Syntax: Communicating Hierarchical Components

This example shows a composition of submodels for a cooperative cruise control system. This model shows a simple adaptive cruise control system, illustrating model-integrated control strategies. A leading car model produces information that is observed with possible flaws by a following car. If the following car detects flaws, it uses a conservative strategy. Otherwise, it tracks the leading car closely.

This example shows a composition of submodels for a cooperative cruise control system.
Static Semantics: Questions to Ask

- Are data types compatible?
- Are units used consistently?
- Are dimensions used consistently?
- Are communication protocols compatible?
- Are components required?
- Are component designs mature?
- .....

I will describe a framework called Ptomas for posing these questions in a domain-specific way and answering them using techniques from type inference.
Static Semantics: Questions to Ask

- Are data types compatible?
- Are units used consistently?
- Are dimensions used consistently?
- Are communication protocols compatible?
- Are components required?
- Are component designs mature?
- ......

I will describe a framework called Ptomas for posing these questions in a domain-specific way and answering them using techniques from type inference.

Dimensions are semantic information associated with the data in the model. A system of such semantic information is called an ontology.
Above is a *lattice* constructed to represent a simple *ontology* in an application domain, such as cruise-control design.
Ptolemy II: Our laboratory for experimenting with modeling, design, and simulation technology

Pthomas: A framework on top of Ptolemy II, allowing analysis of these semantic properties

Types of analysis:
- Inference
- Validity Checking
In our Example Ontology, relations given by basic Calculus

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

\[
\int speed(t) \, dt = position(t)
\]
Applying This Ontology to a Model

- Continuous Director
- DimensionSystemSolver
  - Double click to Resolve Properties
- ConstNonconstSolver
  - Double click to Resolve Properties
- PropertyRemover
  - Double click to Remove Properties
- Integrator
  - Constraint: Const.output == Acceleration
- PropertyLatticeAttribute
- Display
Applying This Ontology to a Model

- Continuous Director
- DimensionSystemSolver
  - Double click to Resolve Properties
  - ConstNonconstSolver
    - Double click to Resolve Properties
- PropertyRemover
  - Double click to Remove Properties

- Integrator
  - Acceleration
  - Speed

- Display
  - Speed

- PropertyLatticeAttribute

- DimensionSystemSolver::Constraint: Const.output == Acceleration
Applying This Ontology to a Model

- Continuous Director
- DimensionSystemSolver
  - Double click to Resolve Properties
- ConstNonconstSolver
  - Double click to Resolve Properties
- PropertyRemover
  - Double click to Remove Properties

- AddSubtract
- Integrator

- DimensionSystemSolver::Constraint: Const.output == Acceleration

PropertyLatticeAttribute
Applying This Ontology to a Model

DimensionSystemSolver
Double click to Resolve Properties

ConstNonconstSolver
Double click to Resolve Properties

PropertyRemover
Double click to Remove Properties

Continuous Director

// Diagram Details

AddSubtract
Integrator

Const
Acceleration

DimensionSystemSolver::Constraint: Const.output == Acceleration

PropertyLatticeAttribute

Edward A. Lee, et al. (UC Berkeley, Bosch)
Applying This Ontology to a Model

This model shows a simple adaptive cruise control system, illustrating model-integrated control strategies. A leading car model produces information that is observed with possible flaws by a following car. If the following car detects flaws, it uses a conservative strategy. Otherwise, it tracks the leading car closely.

Author: Xiaoljun Liu and Edward A. Lee
Applying This Ontology to a Model

This model shows a simple adaptive cruise control system, illustrating model-integrated control strategies. A leading car model produces information that is observed with possible flaws by a following car. If the following car detects flaws, it uses a conservative strategy. Otherwise, it tracks the leading car closely.

Simulate a car that matches the desired speed using feedback control with a specified time constant.

Simulate a wireless network that corrupts the data when the fault input is true.

Simulate a car that attempts to detect faults in communication and act on its behavior.

Author: Xiaoljun Liu and Edward A. Lee
Applying This Ontology to a Model

Car simulator. This model takes as input a desired speed and implements a simple proportional controller with the specified time constant to achieve that speed. It outputs the acceleration, speed, and position of the car.

- initialPosition: 10.0
- initialSpeed: 0.0
- timeConstant: 10.0
- DimensionSystemSolver::constraint: timeConstant >= Time
Applying This Ontology to a Model

Car simulator. This model takes as input a desired speed and implements a simple proportional controller with the specified time constant to achieve that speed. It outputs the acceleration, speed, and position of the car.
Background

- **Lattice**
  A partially ordered set in which every pair of elements has both a least upper bound and a greatest lower bound.

- **Monotonic Function**
  A function $f$ for which
  \[ x_1 \leq x_2 \implies f(x_1) \leq f(x_2) \]
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)

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

is there a satisfying assignment to variables?
Problem Formulation

Problem Statement

Given:

Lattice: \( P \)  
Constants & Variables: \( p_1, p_2, \cdots \in P \)  
Constraints of the form: \( f(p_1, p_2, \cdots) \leq p_n \) (\( f \) monotonic)  

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?

Problem: $|C| \propto |M|$  

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

2. **Actor-specific Constraints**
   - Uses an *adapter pattern* for actors

3. **Instance-specific Constraints**
   - Specified through model annotations
Defining Actor-specific Constraints

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

\[ f_O(p_x) = \begin{cases} 
  \text{Undef.} & \text{if } p_x = \text{Undef.} \\
  \text{Pos.} & \text{if } p_x = \text{Speed} \\
  \text{Speed} & \text{if } p_x = \text{Accel.} \\
  \text{Time} & \text{if } p_x = \text{Unitless} \\
  \text{Error} & \text{otherwise} 
\end{cases} \]
This example illustrates that an ontology can be used to determine in which parts of a model signals vary dynamically.
This example illustrates that an ontology can be used to determine in which parts of a model signals vary dynamically.
Related work

- **Constraint Satisfiability (Rehof and Mogensen)**
  - Linear time algorithm for the monotone function problem

- **Hindley-Milner Type Theory**
  - Sound, incomplete static check of programs before run time.

- **Web Ontology Language (OWL), Eclipse Modeling Framework (EMF), Object Constraint Language (OCL)**
  - **Similarities:** Ontology frameworks (concepts and relationships)
  - **Differences:** Expressiveness, Efficiency, Uniqueness of inference
Open Issues

- **Usability**
  - Defining lattices
  - Giving constraints

- **Handling of conflicts**
  - Is there a “minimal” set of relaxations to the constraints that makes them satisfiable?

- **Extension to infinite lattices**
  - Straightforward in theory, but how to make it usable?

- **Generality of lattice based ontologies**
  - How completely can a unit system be represented?
  - Behavioral types (e.g. using Interface Automata)?
  - ...
Our framework (Pthomas) for analyzing model properties:

1. Customizable for an application domain (lattice and default constraints)
2. Requires minimal user specification (model annotations with specific constraints)
3. Infers unspecified properties
4. Catches and reports design errors
5. Scales up efficiently to large models (leverages Ptolemy II type system implementation by Yuhong Xiong)
The Ptolemy Pteam

See http://chess.eecs.berkeley.edu/pthomas.
Algorithm D (Rehof and Mogensen, 1996)

Pseudocode

\[
\begin{align*}
C_{\text{var}} & \leftarrow \{ \tau \leq A \in C : A \text{ a variable} \} ; \\
C_{\text{const}} & \leftarrow \{ \tau \leq A \in C : A \text{ a constant} \} ; \\
p(\beta) & = \text{Undef. for all variables } \beta ; \\
\text{while } & \text{there are unsatisfied constraints in } C_{\text{var}} \text{ do} \\
& \quad \text{Let } \tau \leq \beta \text{ be one such constraint} ; \\
& \quad \beta \leftarrow \beta \lor \tau ; \\
\text{end} \\
\text{if } & \text{there are unsatisfied constraints in } C_{\text{const}} \text{ then} \\
& \quad \text{Fail: There is no solution} ; \\
\text{end}
\end{align*}
\]

For a finite lattice, this algorithm takes \( O(\text{height}(L) \times |C|) \).
Conflicts

- What is a conflict?
  - Unsatisfiable constraints

- How is it detected?

\[
C_{const} \leftarrow \{ \tau \leq A \in C : A \text{ a constant} \} ;
\]
\[
\ldots
\]
\[
\text{if there are unsatisfied constraints in } C_{const} \text{ then}
\]
\[
\text{Fail: There is no solution ;}
\]
\[
\text{end}
\]