Ensuring Correct Composition of Components using Lattice-based Ontologies

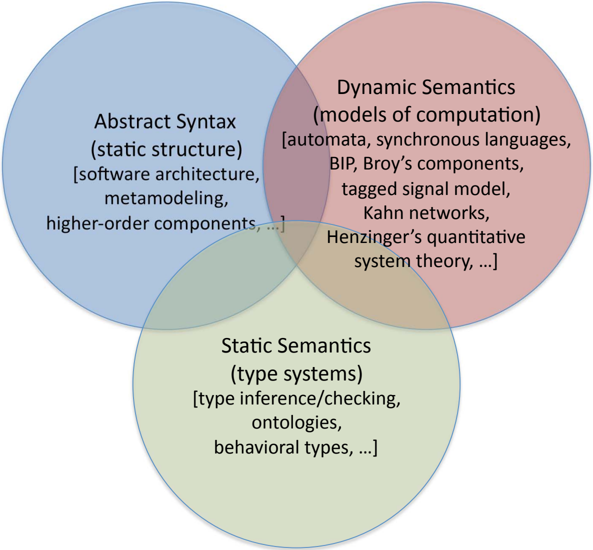
Ben Lickly[1] Man-Kit Leung[1] Thomas Mandl[2] Edward A. Lee[1]
Elizabeth Latronico[2] Charles Shelton[2] Stavros Tripakis[1]

[1]University of California, Berkeley

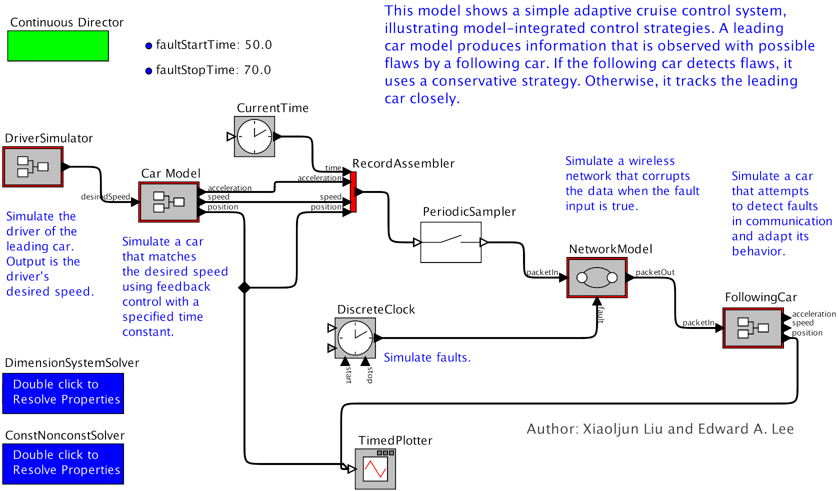
[2]Bosch Research, LLC

WFCD - Foundations and Applications of Component-based Design
in Conjunction with ES Week
Grenoble, France, Oct. 11, 2009

Taxonomy of Modeling Issues



Abstract Syntax: Communicating Hierarchical Components



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.

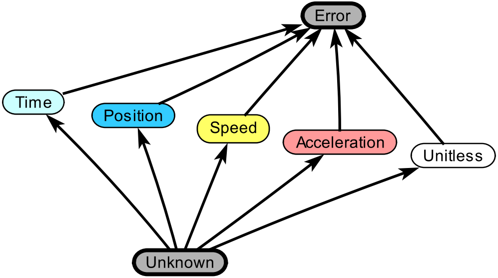
Static Semantics: Questions to Ask

- 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*.

Static Semantics: Defining an Ontology



Above is a *lattice* constructed to represent a simple *ontology* in an application domain, such as cruise-control design.

Approach

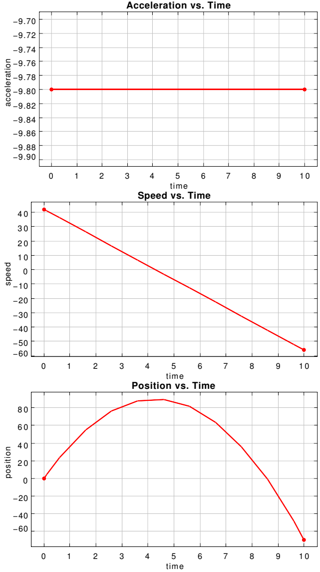
Ptolemy II: Our laboratory for experimenting with modeling, design, and simulation technology

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

Types of analysis:

- Inference
- Validity Checking

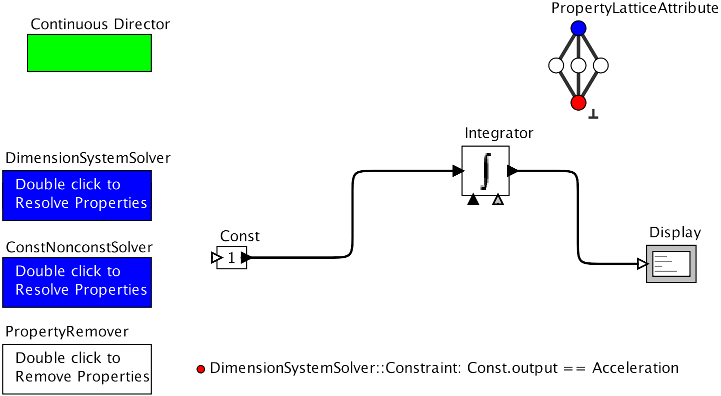
In our Example Ontology, relations given by basic Calculus



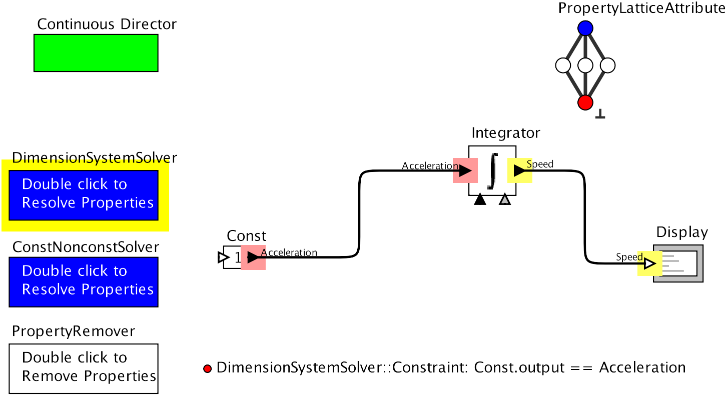
$$\int \textcolor{red}{acceleration}(t)\,dt = \textcolor{orange}{speed}(t)$$

$$\int \textcolor{orange}{speed}(t)\,dt = \textcolor{blue}{position}(t)$$

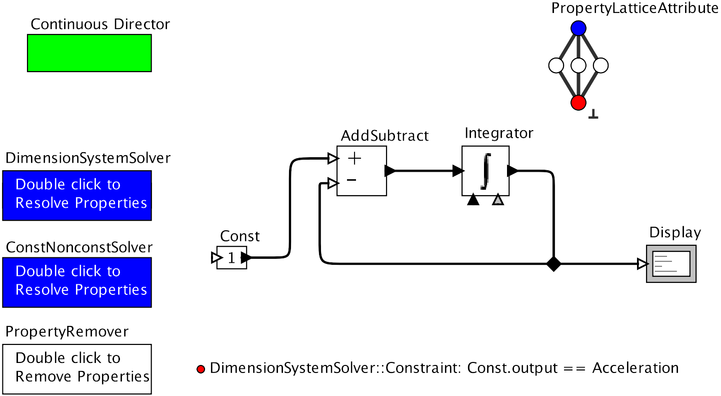
Applying This Ontology to a Model



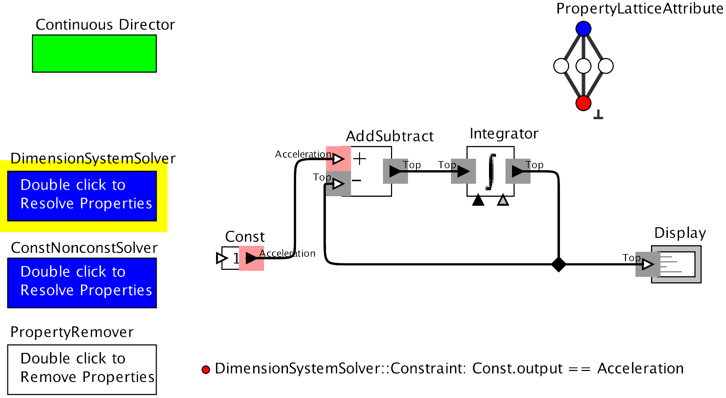
Applying This Ontology to a Model



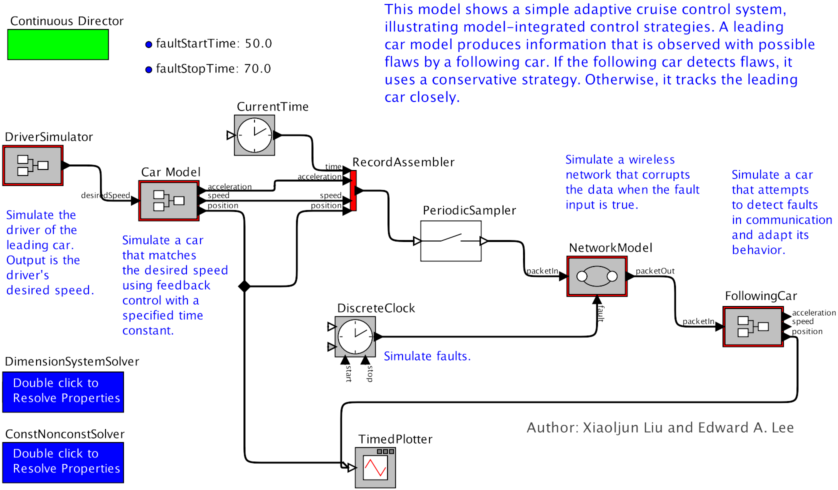
Applying This Ontology to a Model



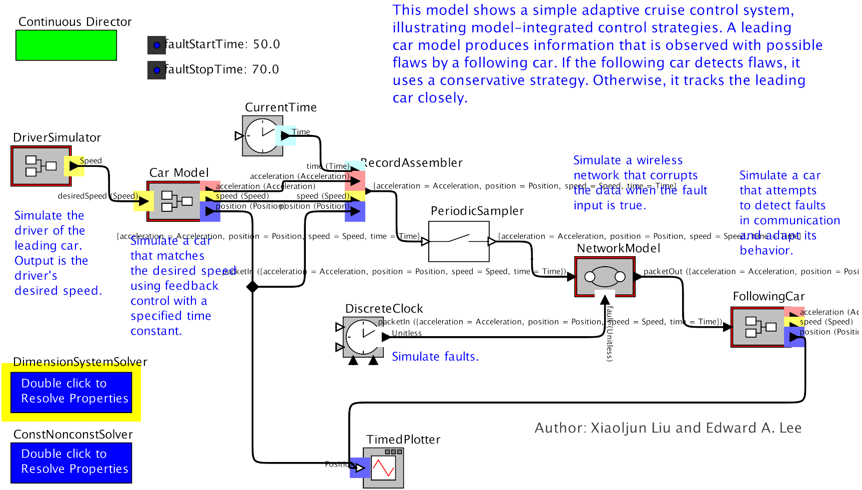
Applying This Ontology to a Model



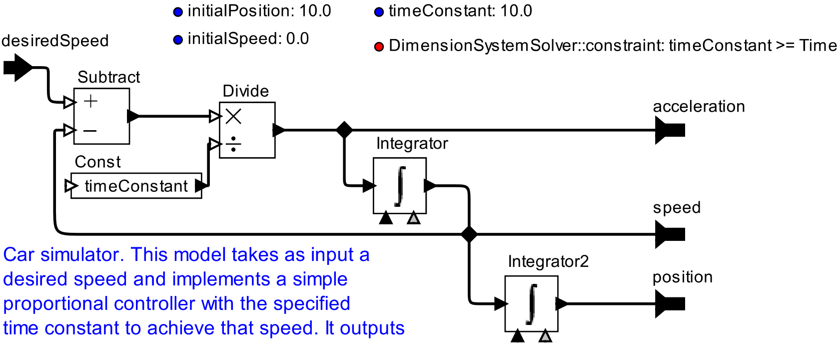
Applying This Ontology to a Model



Applying This Ontology to a Model

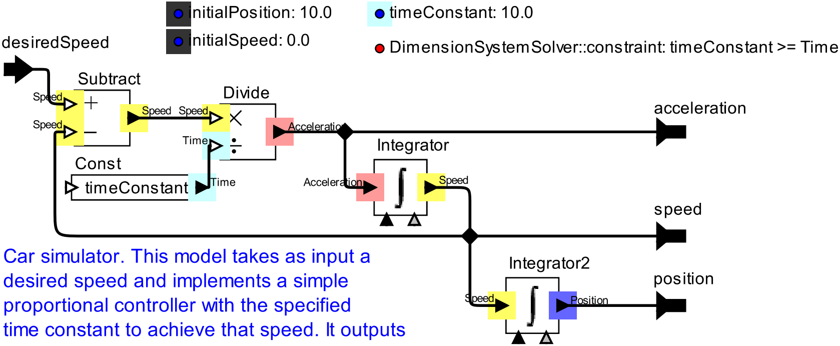


Applying This Ontology to a Model



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.

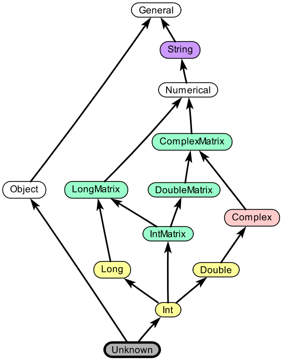
Applying This Ontology to a Model



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
  $\vec{x_1} \leq \vec{x_2} \implies f(\vec{x_1}) \leq f(\vec{x_2})$



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, $\cdots$.

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:

$$\begin{aligned} \text{Lattice:} \quad & P && (1) \\ \text{Constants \& Variables:} \quad & p_1, p_2, \cdots \in P && (2) \\ \text{Constraints of the form:} \quad & f(p_1, p_2, \cdots) \leq p_n \qquad (f \text{ monotonic}) && (3) \end{aligned}$$

is there a satisfying assignment to variables?

Problem Statement

Given:

$$\text{Lattice:} \quad P \tag{1}$$

$$\text{Constants \& Variables:} \quad p_1, p_2, \cdots \in P \tag{2}$$

$$\text{Constraints of the form:} \quad f(p_1, p_2, \cdots) \leq p_n \qquad (f \text{ monotonic}) \tag{3}$$

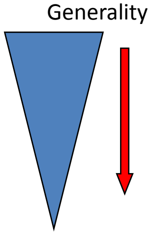
is there a satisfying assignment to variables?

This problem has a linear time algorithm!
(Rehof and Mogensen, 1996)

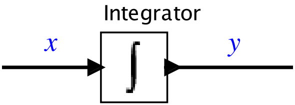
How to Make this Usable in Practice?

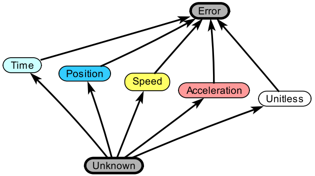
Problem: $|C| \propto |M|$

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

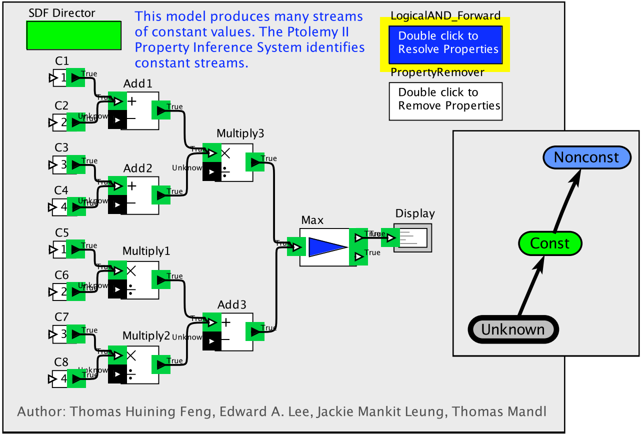




Component	Elements	Constraints
Integrator	input x, output y	$f_I(p_y) \leq p_x$ $f_O(p_x) \leq p_y$

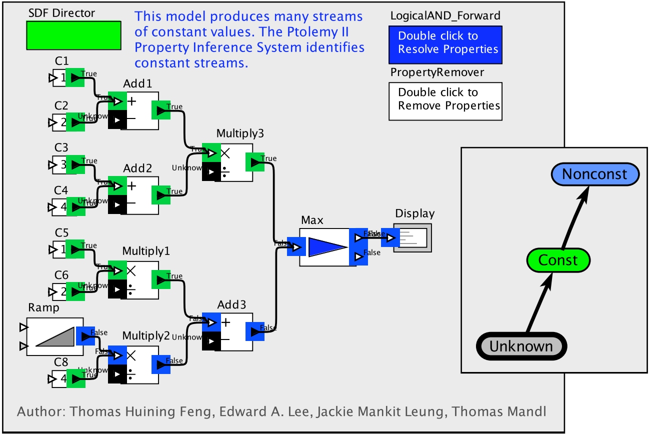
$$f_I(p_y) = \begin{cases} \text{Undef.} & \text{if } p_y = \text{Undef.} \\ \text{Speed} & \text{if } p_y = \text{Pos.} \\ \text{Accel.} & \text{if } p_y = \text{Speed} \\ \text{Unitless} & \text{if } p_y = \text{Time} \\ \text{Error} & \text{otherwise} \end{cases} \qquad f_O(p_x) = \begin{cases} \text{Undef.}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

Another Example



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

Another Example



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)?
  - ...

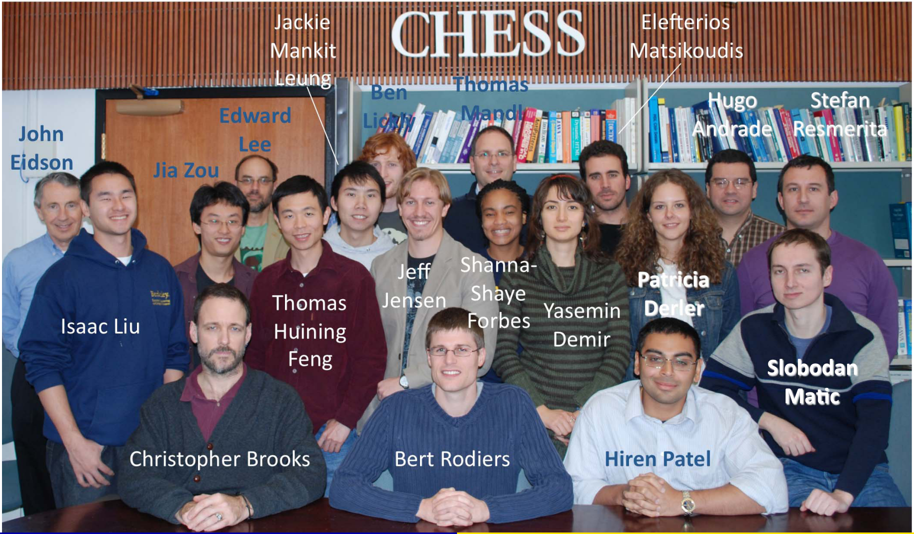
Summary

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

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

For a finite lattice, this algorithm takes $O(height(L) \times |C|)$.

Conflicts

- What is a conflict?
  - Unsatisfiable constraints
- How is it detected?

$C_{const} \leftarrow \{\tau \leq A \in C : A \text{ a constant}\}$;

...

if *there are unsatisfied constraints in* C_{const} **then**

Fail: There is no solution ;

end