Static Analysis using the Ptolemy II Ontologies Package

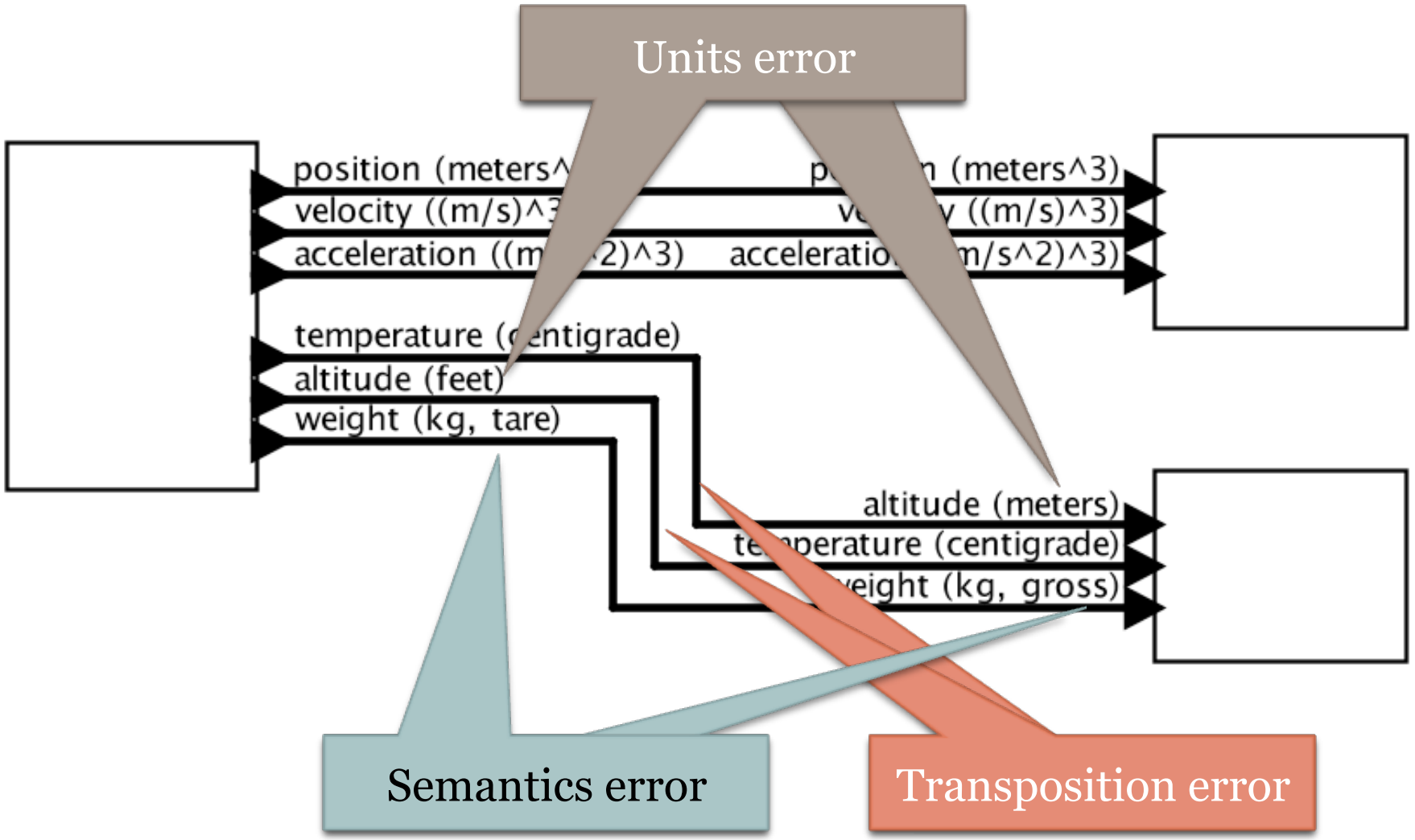
Charles Shelton, Elizabeth Latronico (Bosch Research and Technology Center)
Ben Lickly, Edward Lee (UC Berkeley)

Ptolemy Mini-Conference, February 16, 2011

Motivation

- Cars are networked software systems
  - Up to 70 Electronic Control Units
  - Software crucial for many features
    - Electronic stability control
    - Parking assist
    - Emissions control
    - Engine Start/Stop
    - Active and passive safety
  - ***Bosch makes all of these systems for auto manufacturers***
- How can we manage increasing complexity and interconnectedness of software models?
- Analysis approaches promising, but hand-annotation has drawbacks
  - Time intensive to develop and maintain
  - People are inconsistent, make errors
  - Repeat for every composition

Examples of Model Construction Errors



Static Analysis Using Ontologies

- An ***Ontology*** consists of:
  - A set of ***Concepts***
  - ***Relationships*** between those concepts

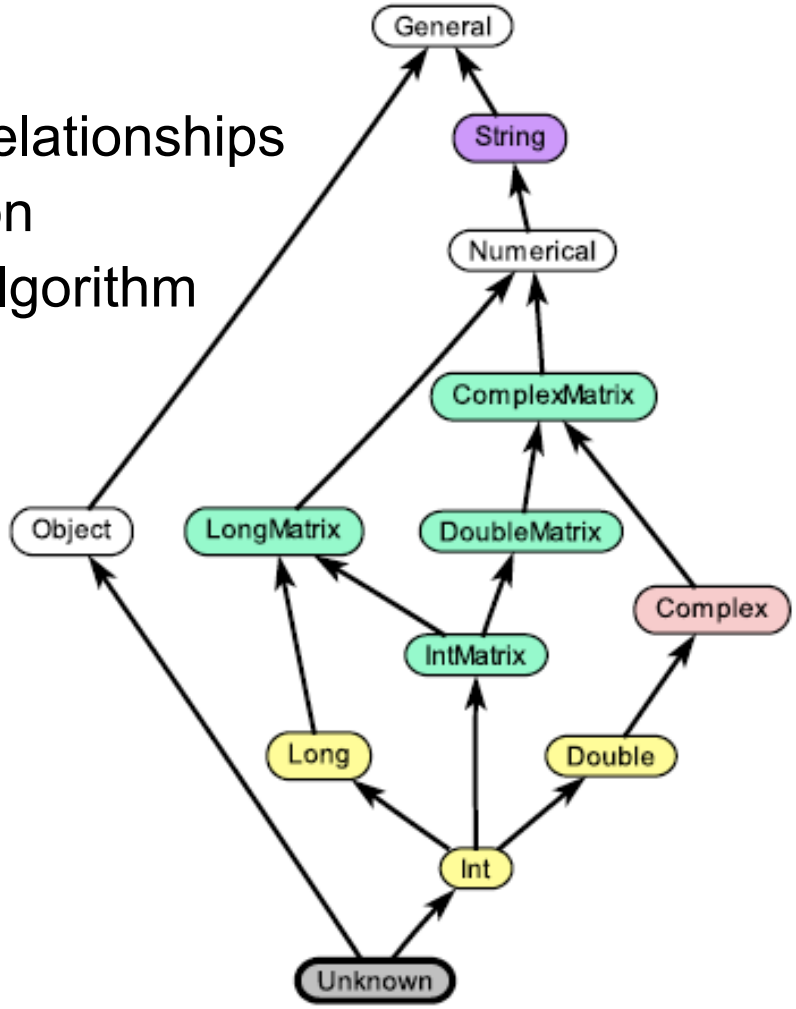
- Ontologies are used for representation of semantic information
  - General ontology frameworks (eg. OWL) focus on expressiveness
  - Arbitrary ontologies represent complex relationships as a graph

- Restrict Ptolemy ontologies to ***lattice*** graph structure
  - Lattice elements form a complete partial order
  - Existing scalable analysis algorithms
  - Existing work from compiler static analysis

Ontology Example: Ptolemy Type System

- Ptolemy type system implementation
  - Types organized in a lattice
  - Edges represent “can be converted to” relationships
  - Automatic type inference and propagation
    - Rehof-Mogensen constraint solving algorithm

- Users define type constraints in their actors
  - eg. An actor’s output port type must be ***“greater than or equal to”*** *(higher in the lattice)* the input port type
  - Connections between actors imply that the sink type **≥** the source type



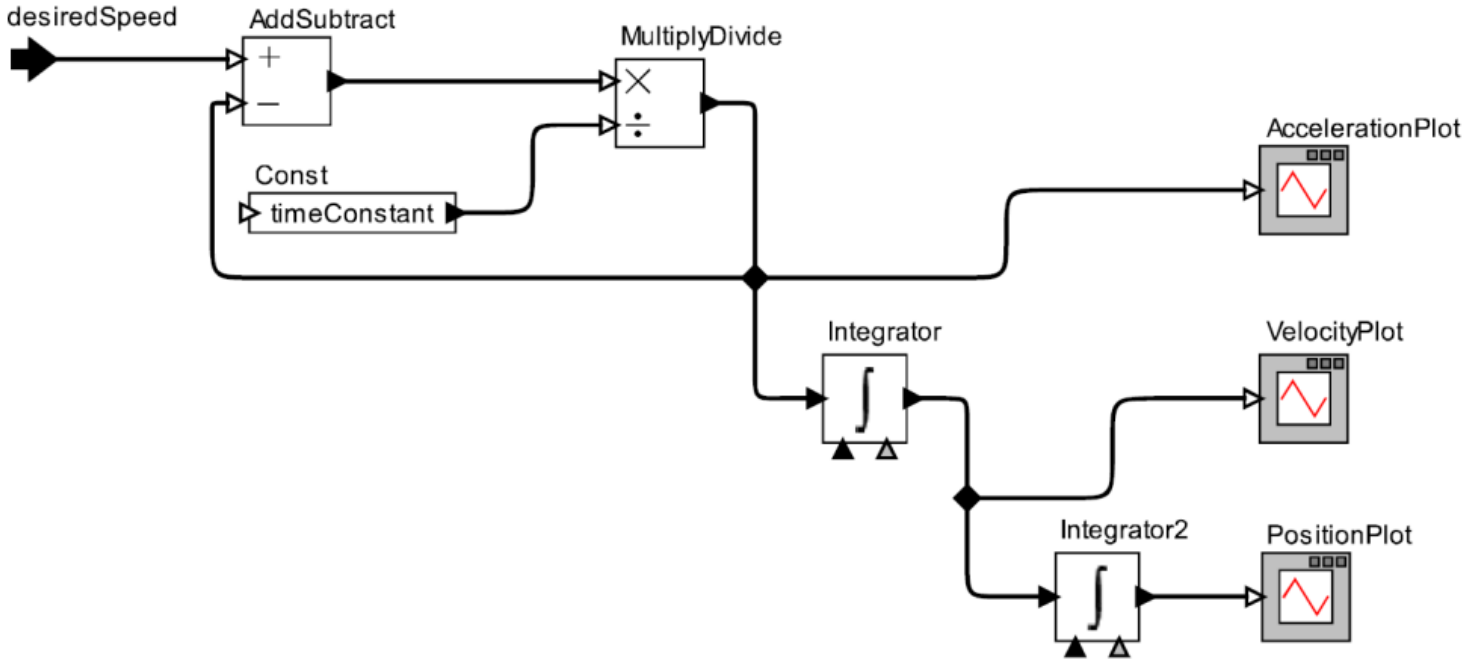
Ptolemy Ontologies Framework

- Ontologies package generalizes the Ptolemy type system framework
  - Users can define their own ontology
  - Must also define the rules that determine ontology concept resolution
    - Constraints between model elements
    - Constraints between actor input and output ports
  - Reuses existing Ptolemy code

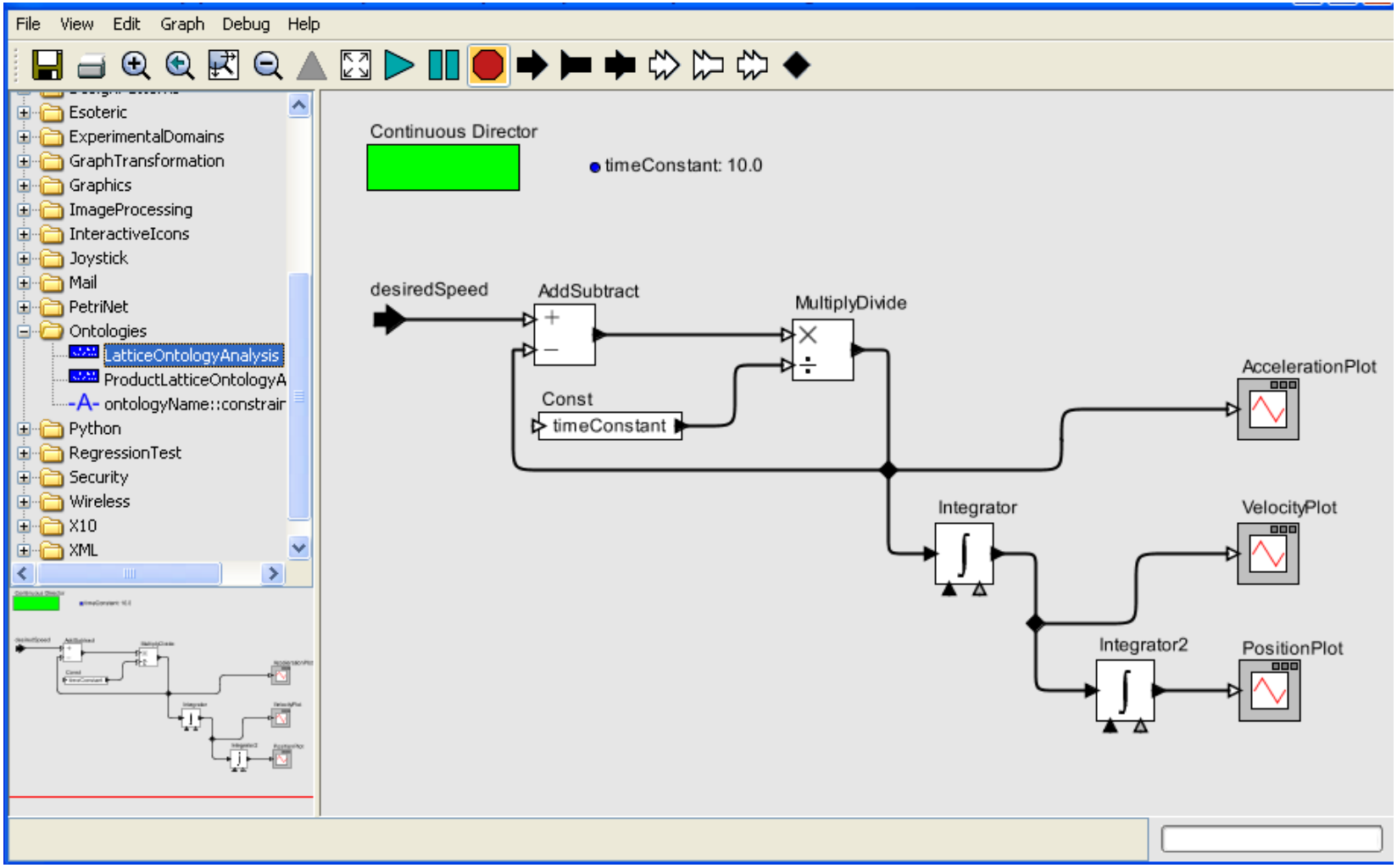
- Constraints are specified as inequalities between concepts assigned to each model element
  - $c_{output} \geq c_{input}$
  - $c_{output} \geq f(c_{input})$ where f is a monotonic function in the ontology domain ($c_a \geq c_b$ ***implies*** $f(c_a) \geq f(c_b)$)

Demo: Dimensional Analysis

- Use the Ontology static analysis to infer dimensional properties
  - Position, Velocity, Acceleration, Time
- Ptolemy Model Example: Simple Car Dynamics Model
  - There is an error in this model that leads to incorrect results

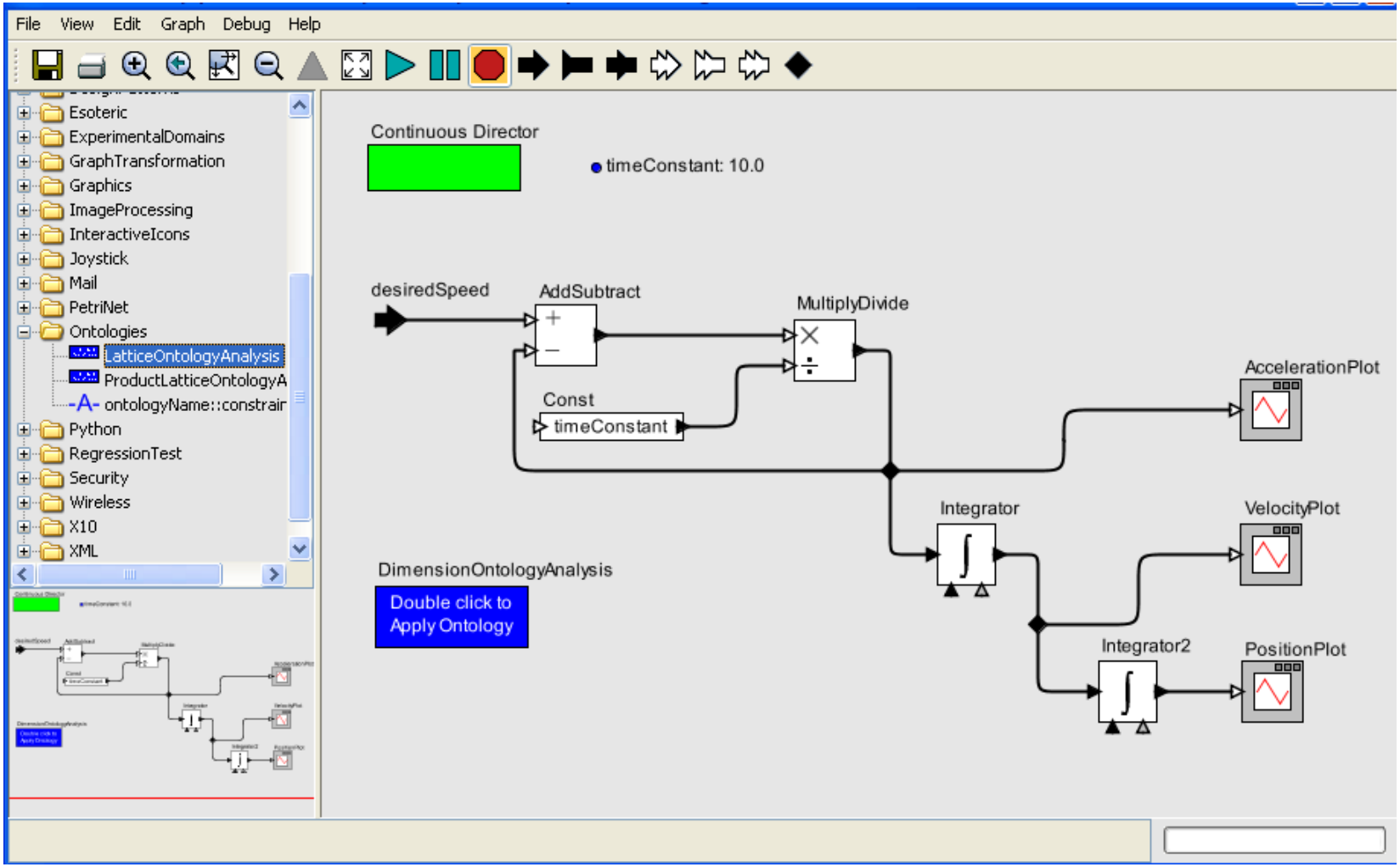


Step 1: Drag in a Lattice Ontology Solver

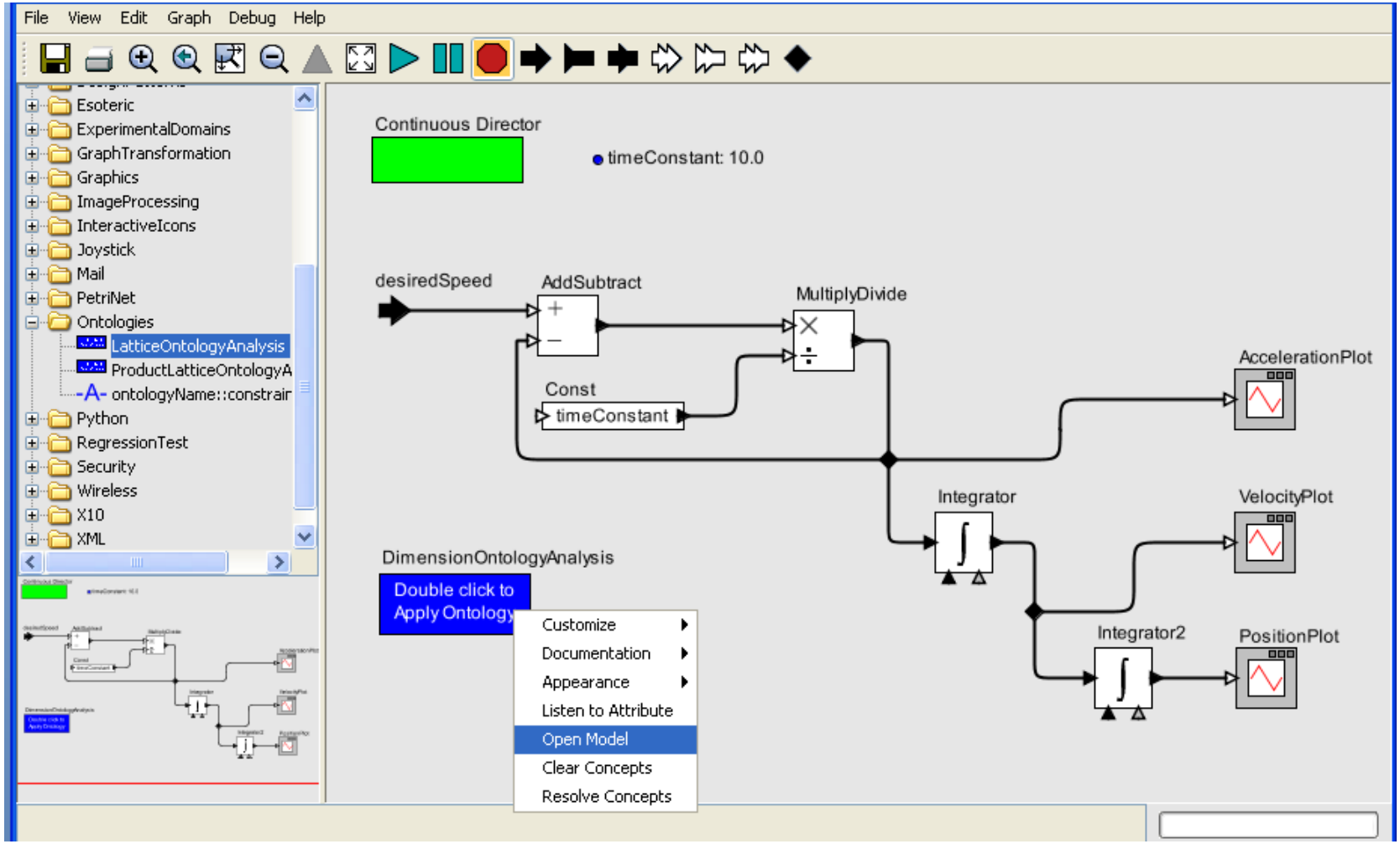


Charles Shelton, Elizabeth Latronico (Bosch), Ben Lickly, Edward Lee (UC Berkeley) | 2/16/2011 |

Step 1: Drag in a Lattice Ontology Solver

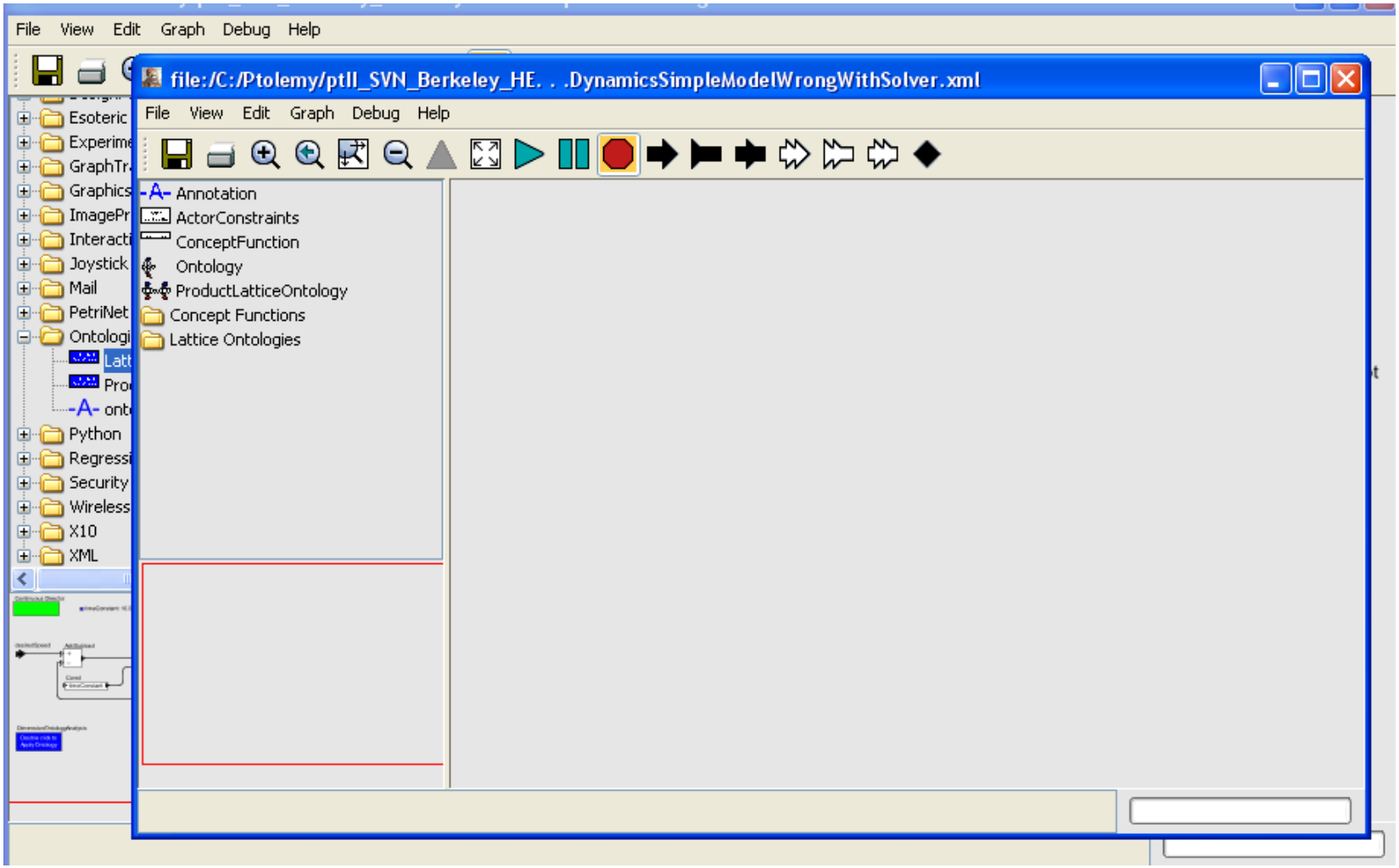


Step 2: Open the Solver Model



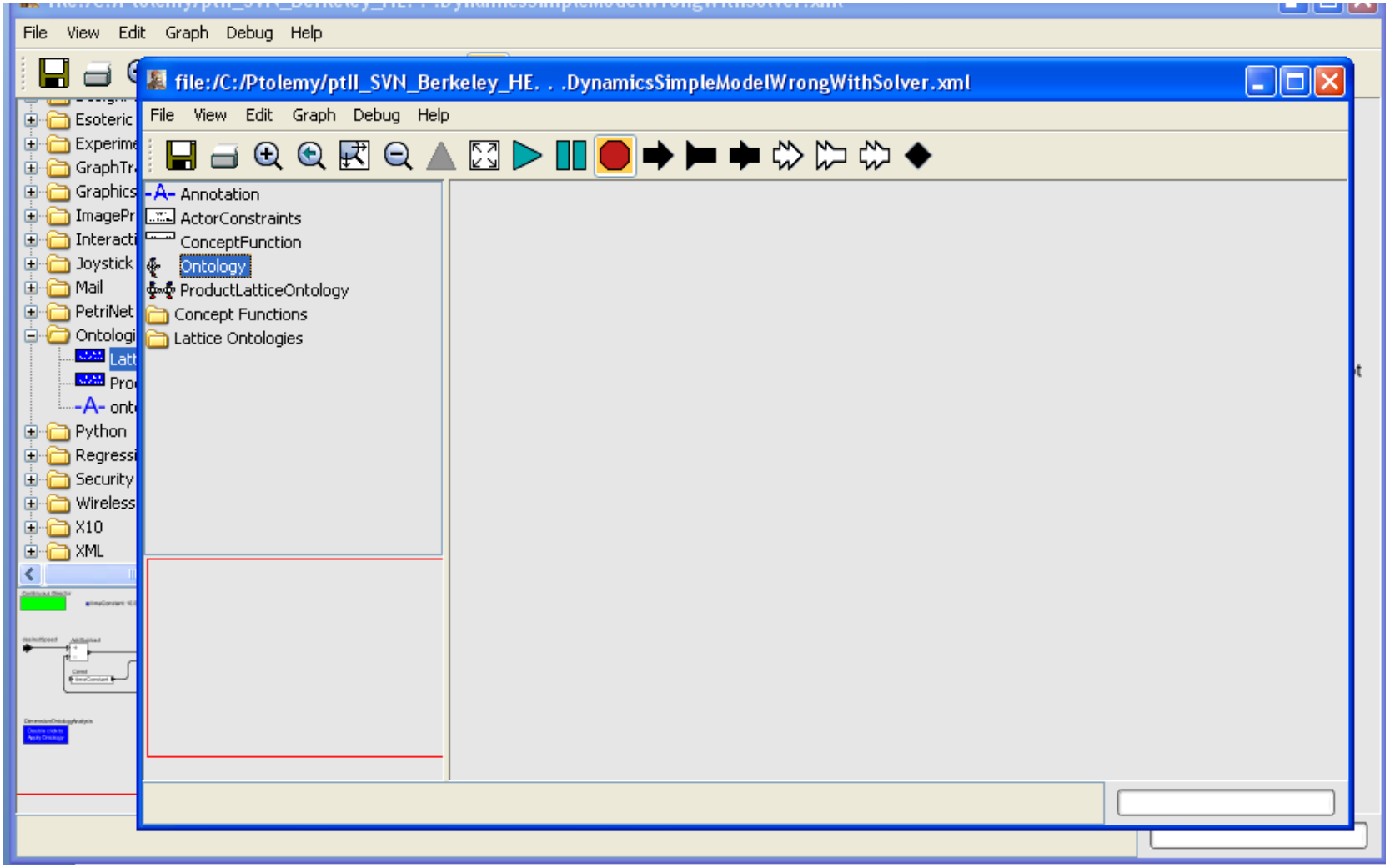
Charles Shelton, Elizabeth Latronico (Bosch), Ben Lickly, Edward Lee (UC Berkeley) | 2/16/2011 |

Step 2: Open the Solver Model

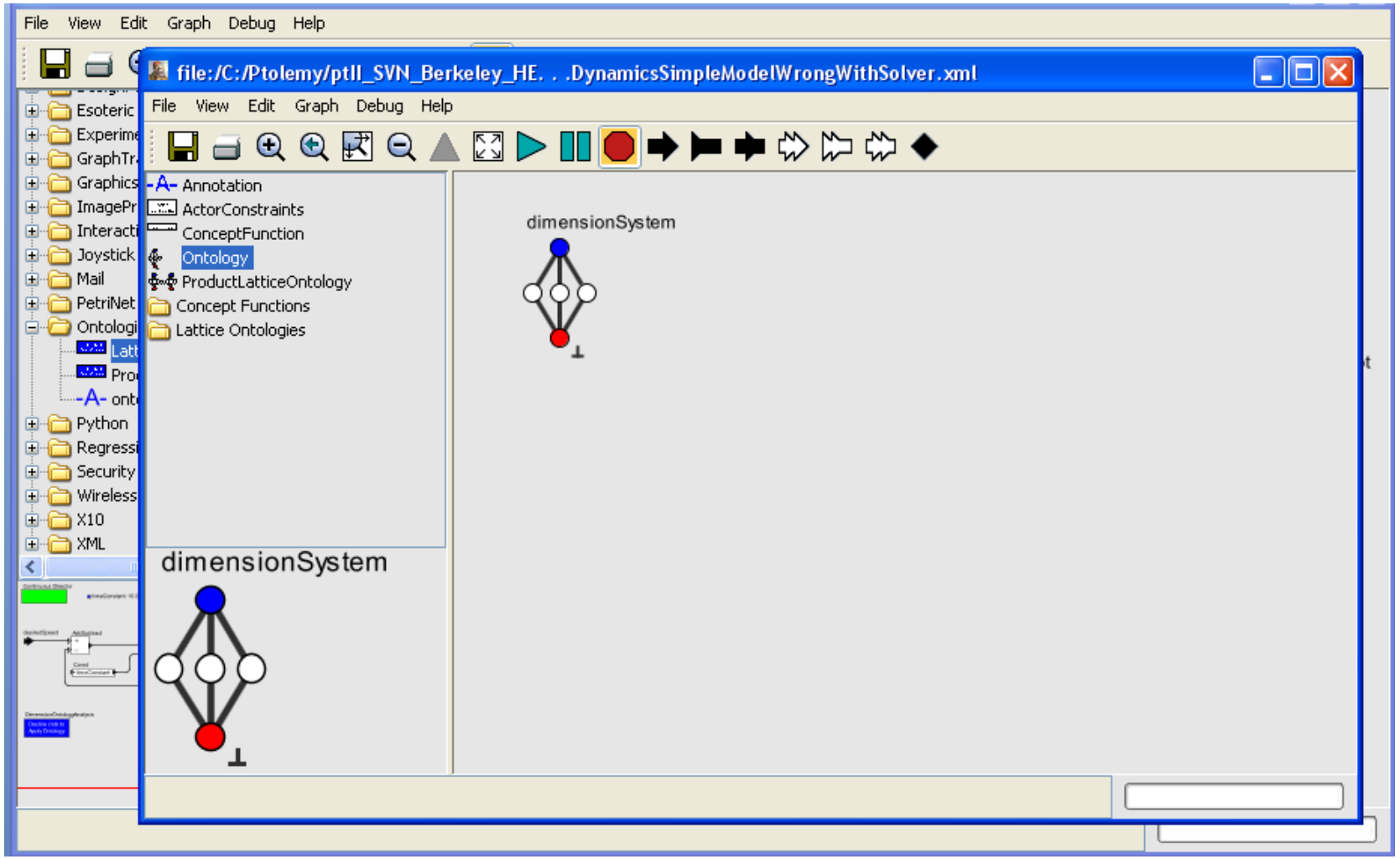


Charles Shelton, Elizabeth Latronico (Bosch), Ben Lickly, Edward Lee (UC Berkeley) | 2/16/2011 |

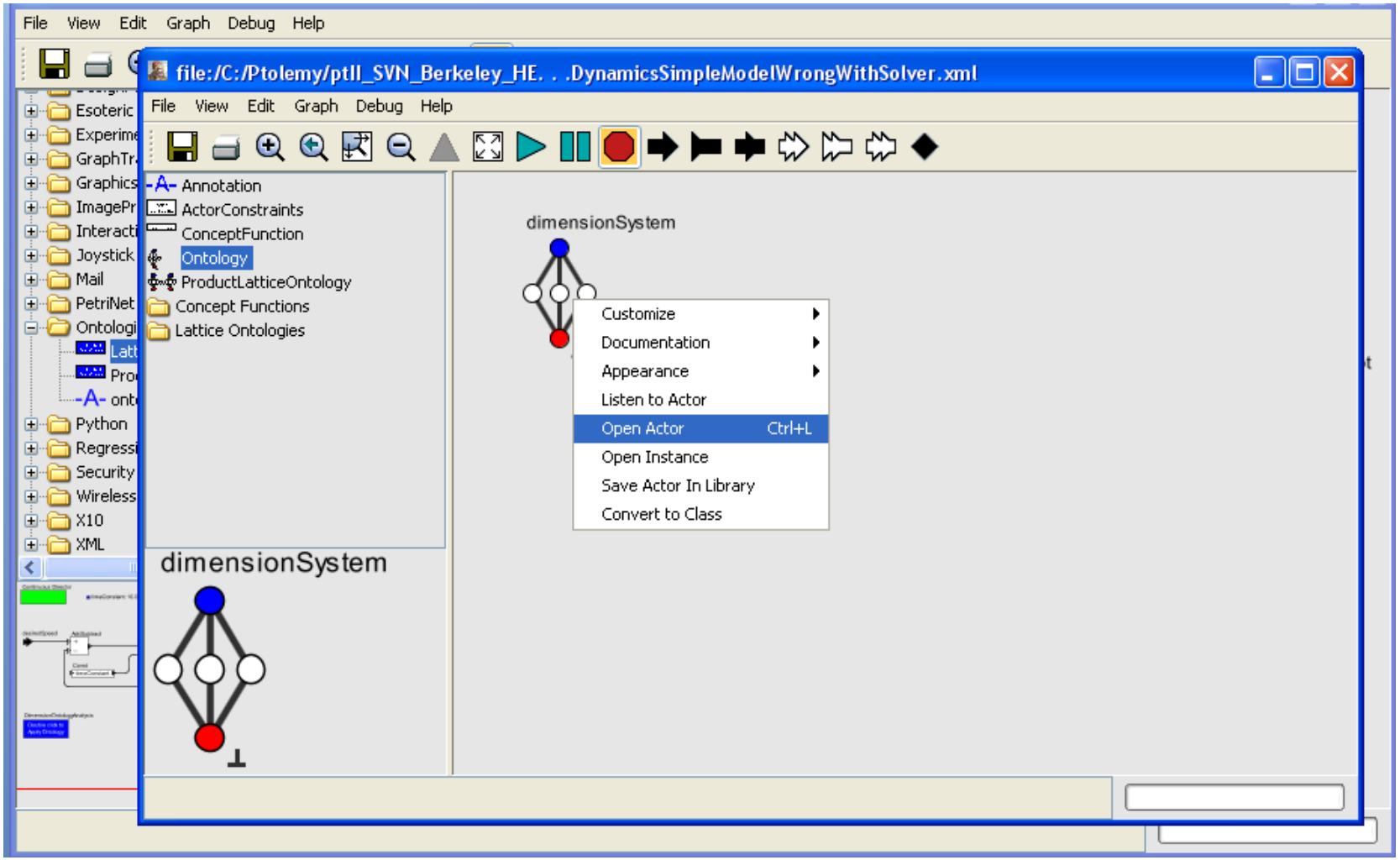
Step 3: Drag an Ontology into the Solver Model



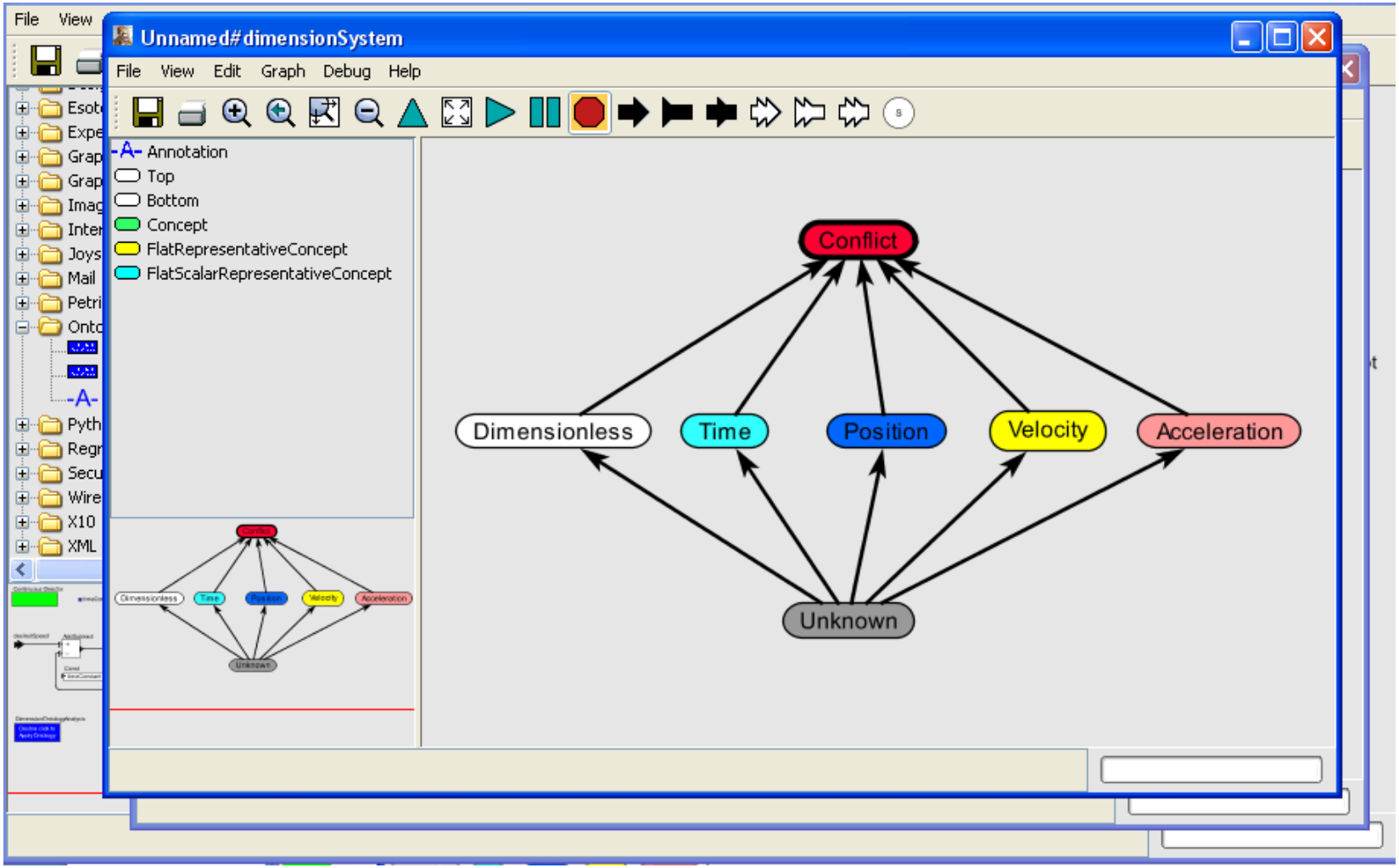
Step 3: Drag an Ontology into the Solver Model



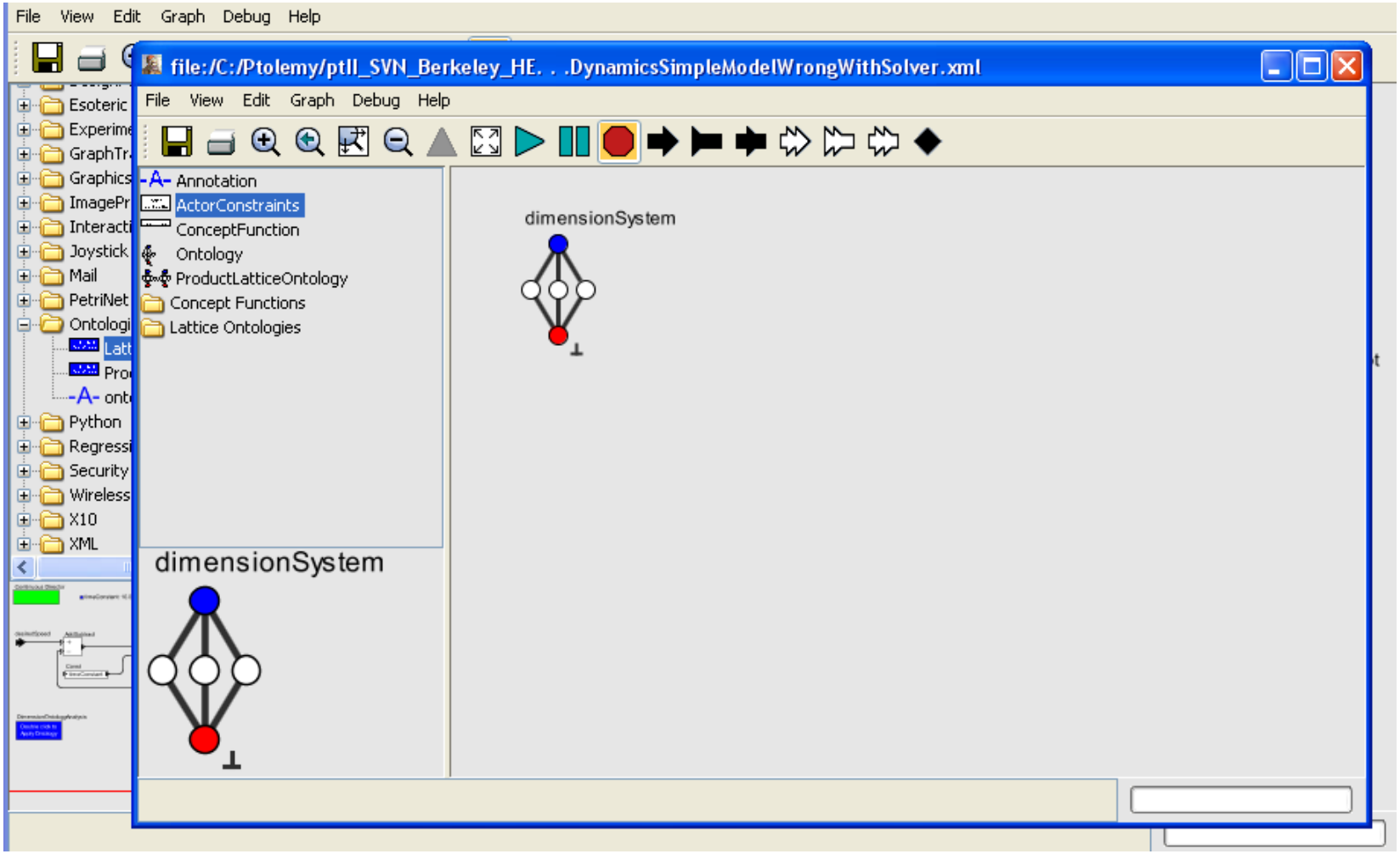
Step 4: Create the Dimensional Analysis Ontology



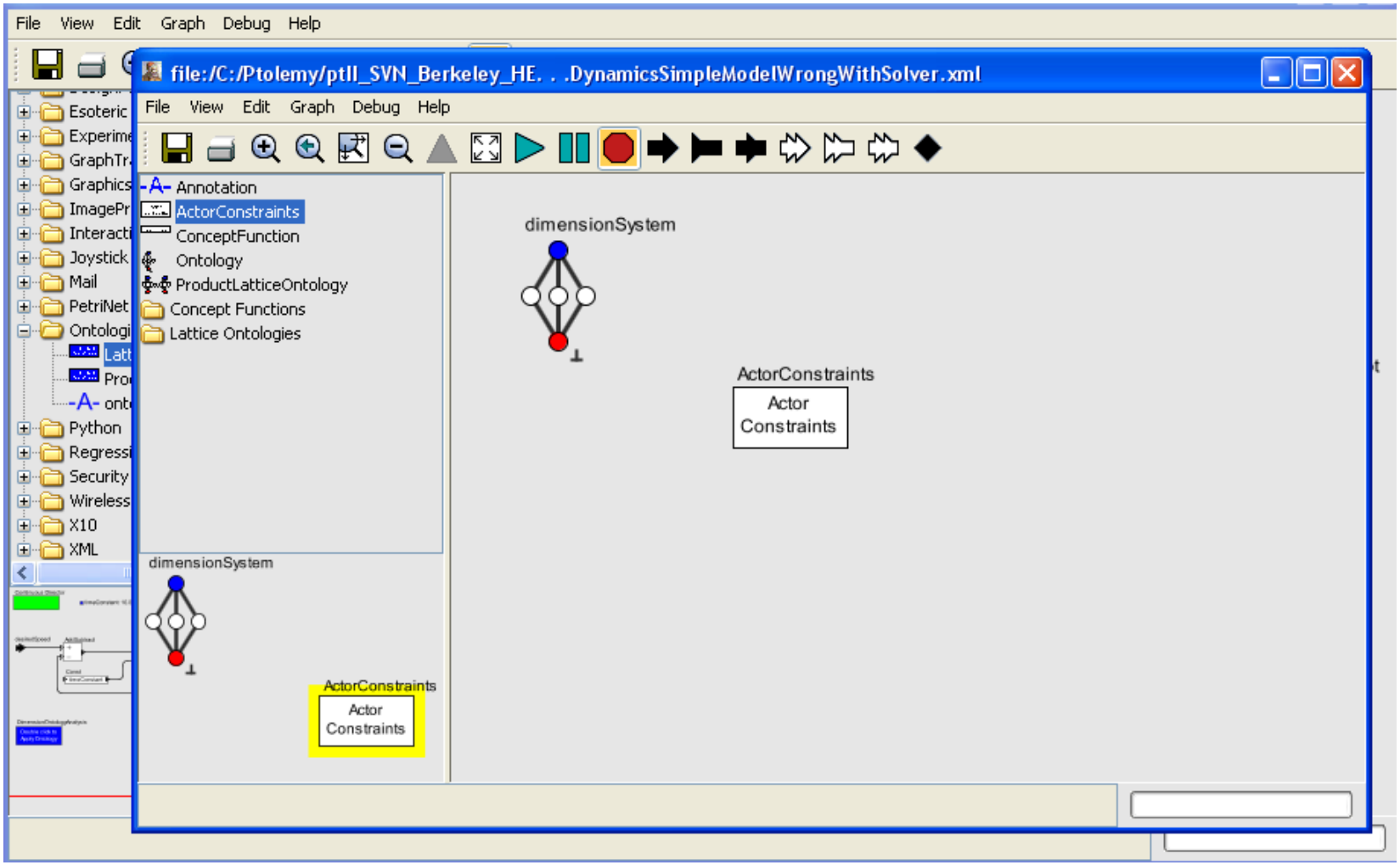
Step 4: Create the Dimensional Analysis Ontology



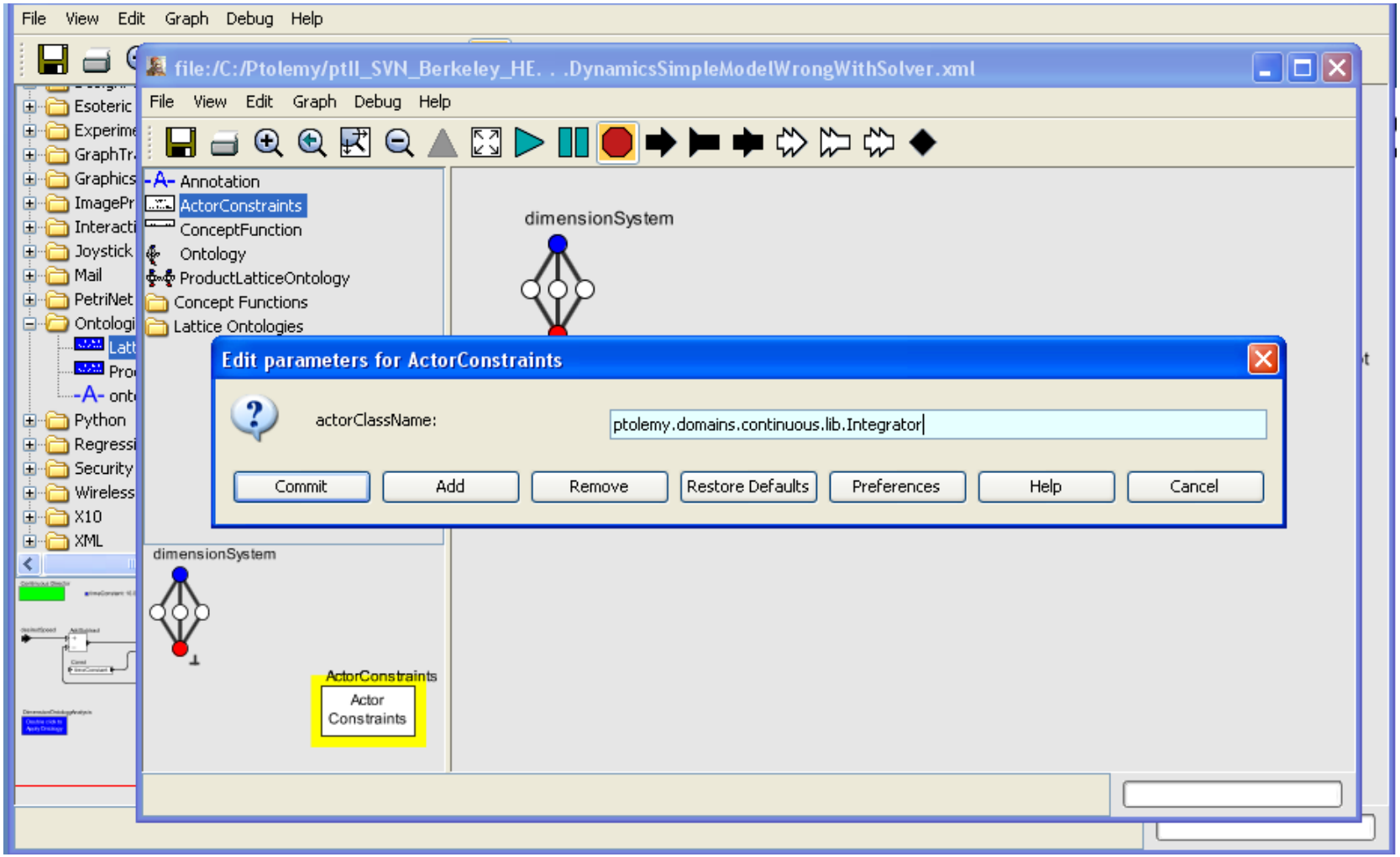
Step 5: Add Actor Constraints to the Solver Model



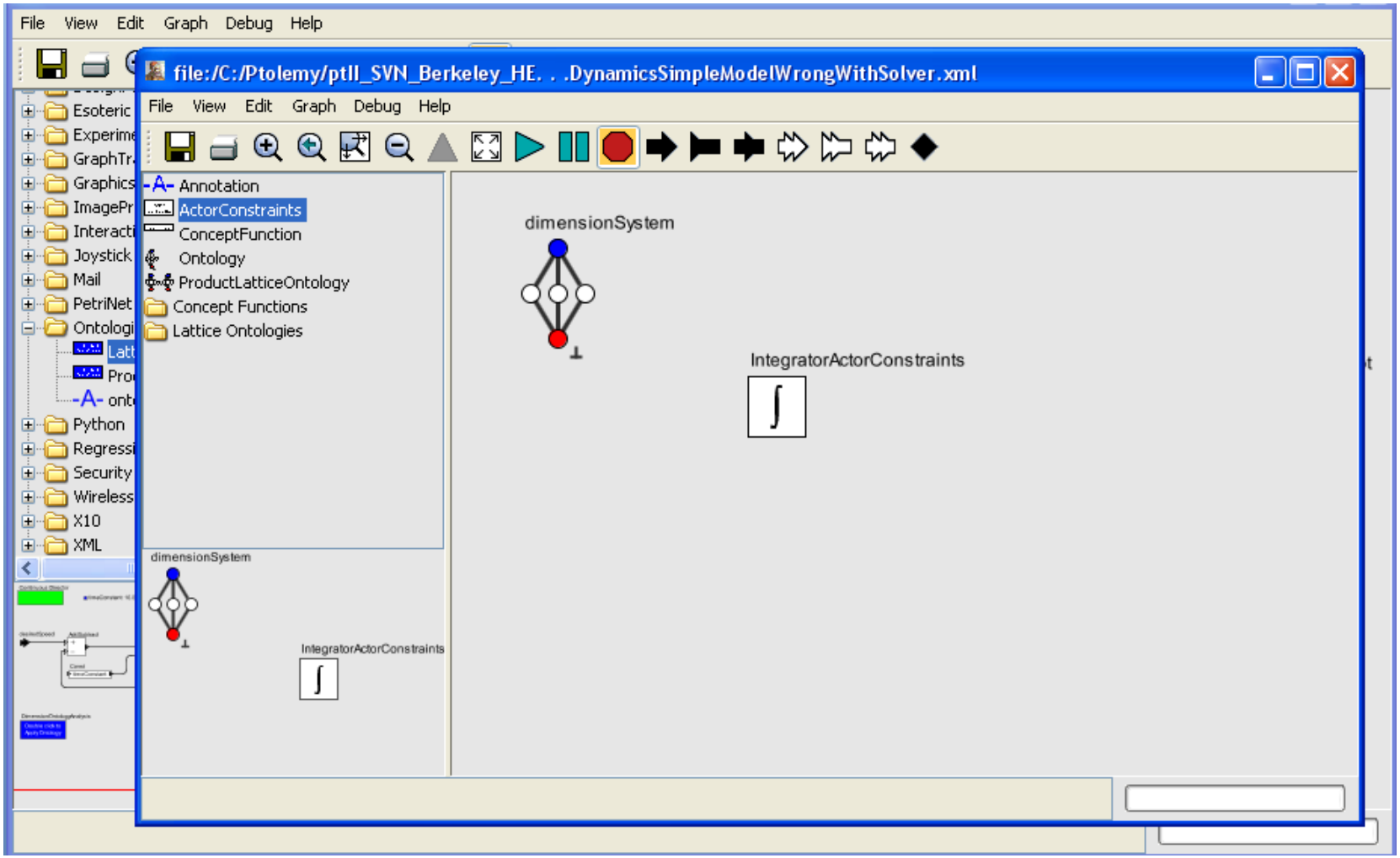
Step 5: Add Actor Constraints to the Solver Model



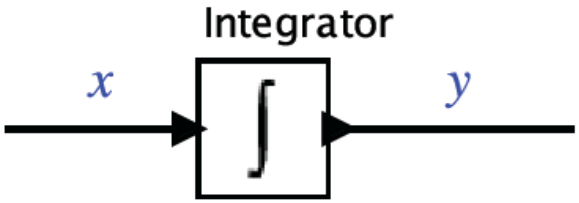
Step 5: Add Actor Constraints to the Solver Model

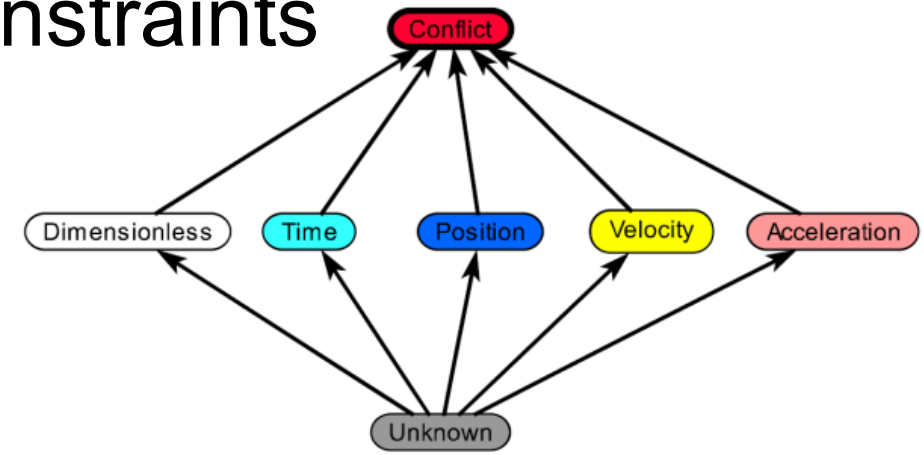


Step 5: Add Actor Constraints to the Solver Model



Defining Actor-Specific Constraints



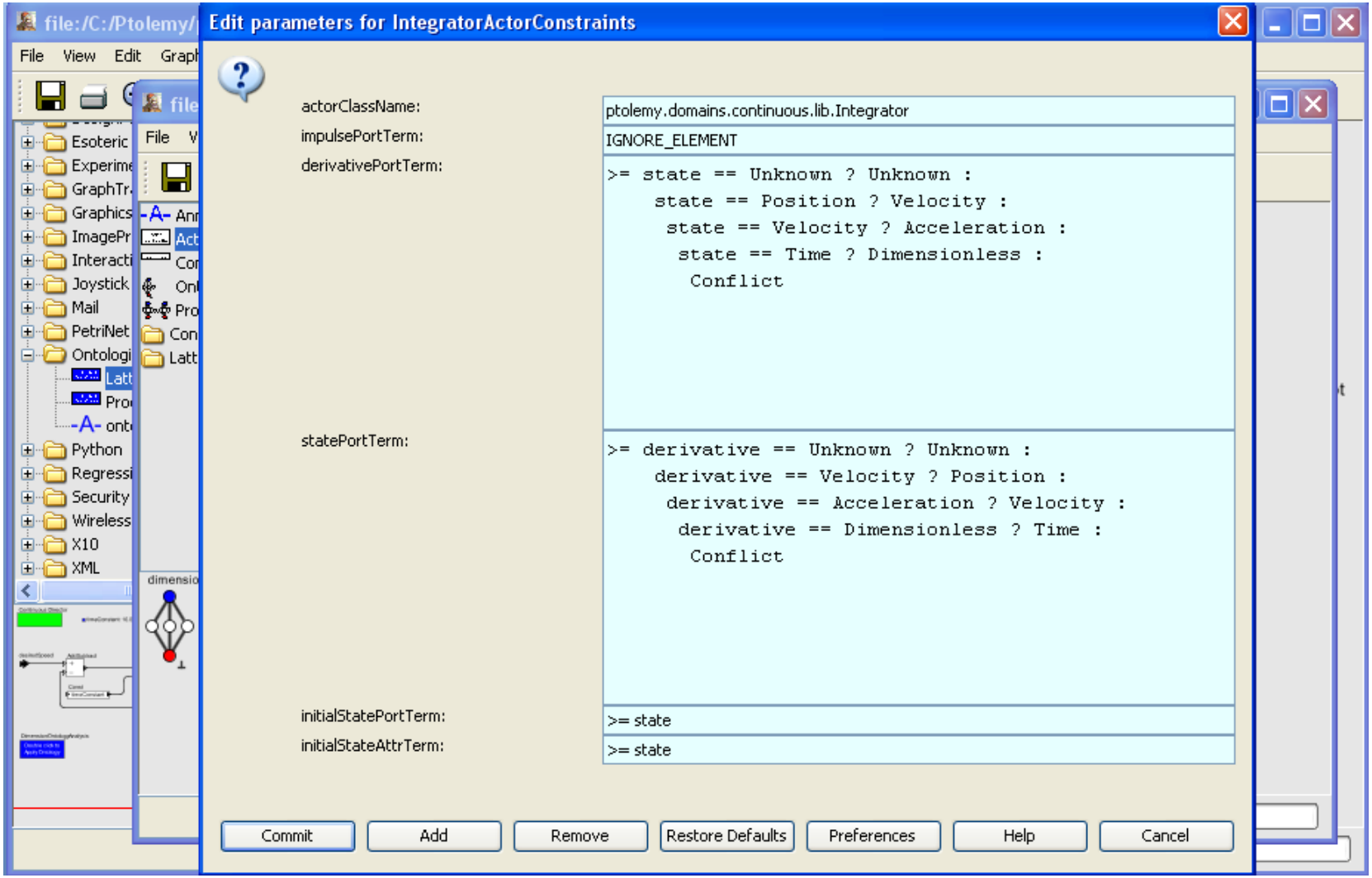


Actor	Elements	Constraints
Integrator	input port ***derivative*** (x), output port ***state*** (y)	$c_x \geq f_I(c_y)$ $c_y \geq f_O(c_x)$

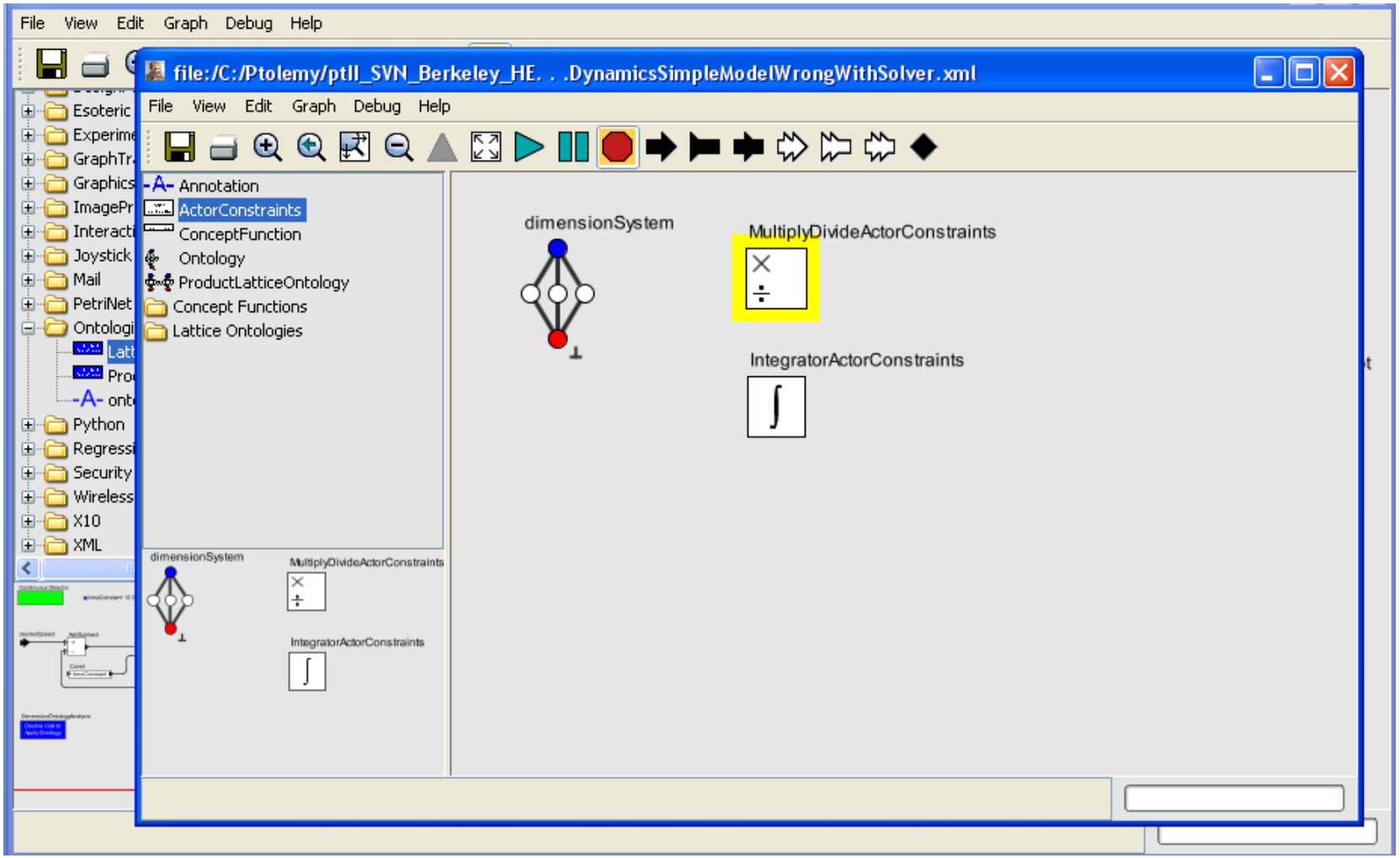
$$f_I(c_y) = \begin{cases} \text{Unknown} & \text{If } c_y = \text{Unknown} \\ \text{Velocity} & \text{If } c_y = \text{Position} \\ \text{Acceleration} & \text{If } c_y = \text{Velocity} \\ \text{Dimensionless} & \text{If } c_y = \text{Time} \\ \text{Conflict} & \text{Otherwise} \end{cases}$$

$$f_O(c_x) = \begin{cases} \text{Unknown} & \text{If } c_x = \text{Unknown} \\ \text{Position} & \text{If } c_x = \text{Velocity} \\ \text{Velocity} & \text{If } c_x = \text{Acceleration} \\ \text{Time} & \text{If } c_x = \text{Dimensionless} \\ \text{Conflict} & \text{Otherwise} \end{cases}$$

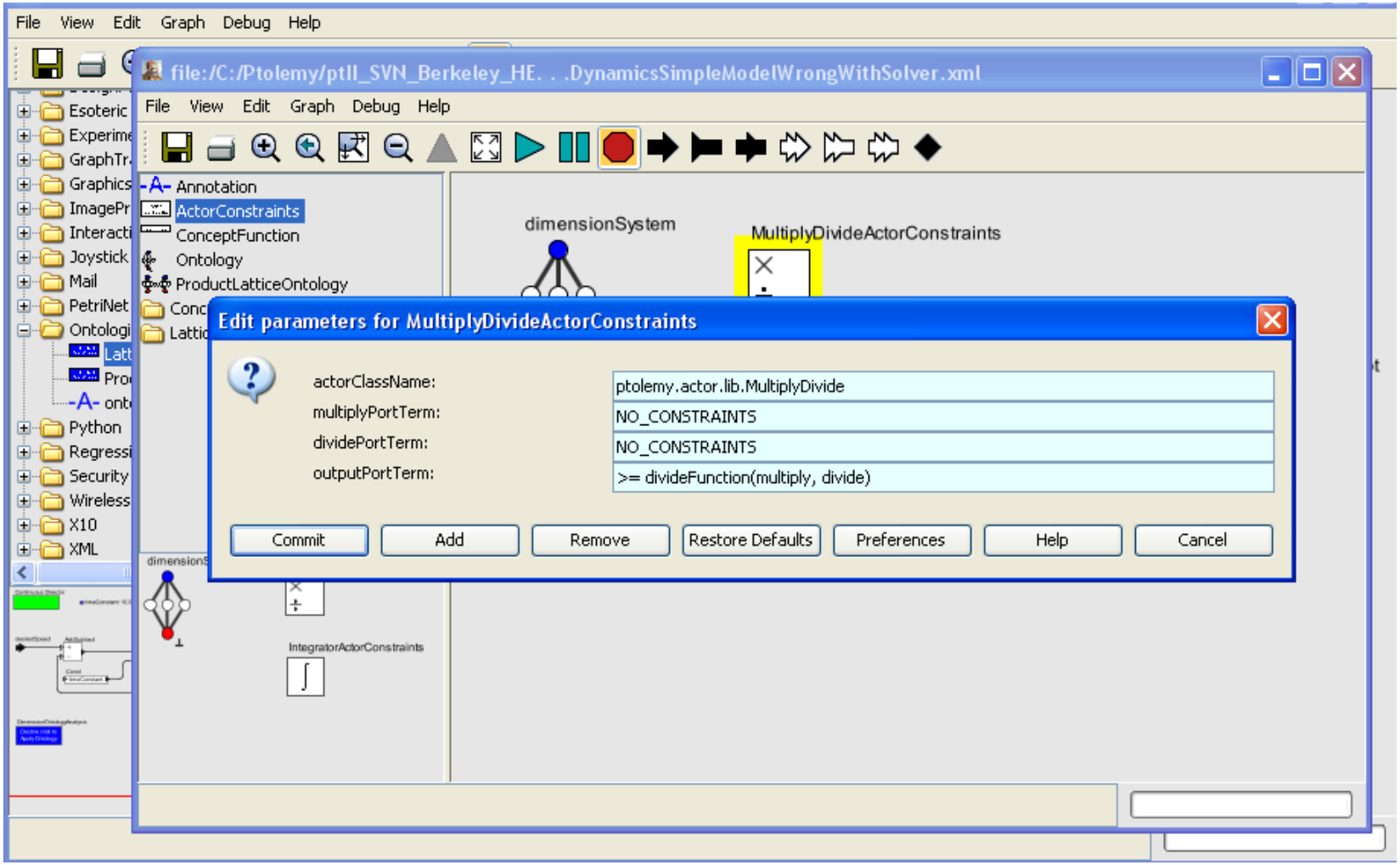
Step 5: Add Actor Constraints to the Solver Model



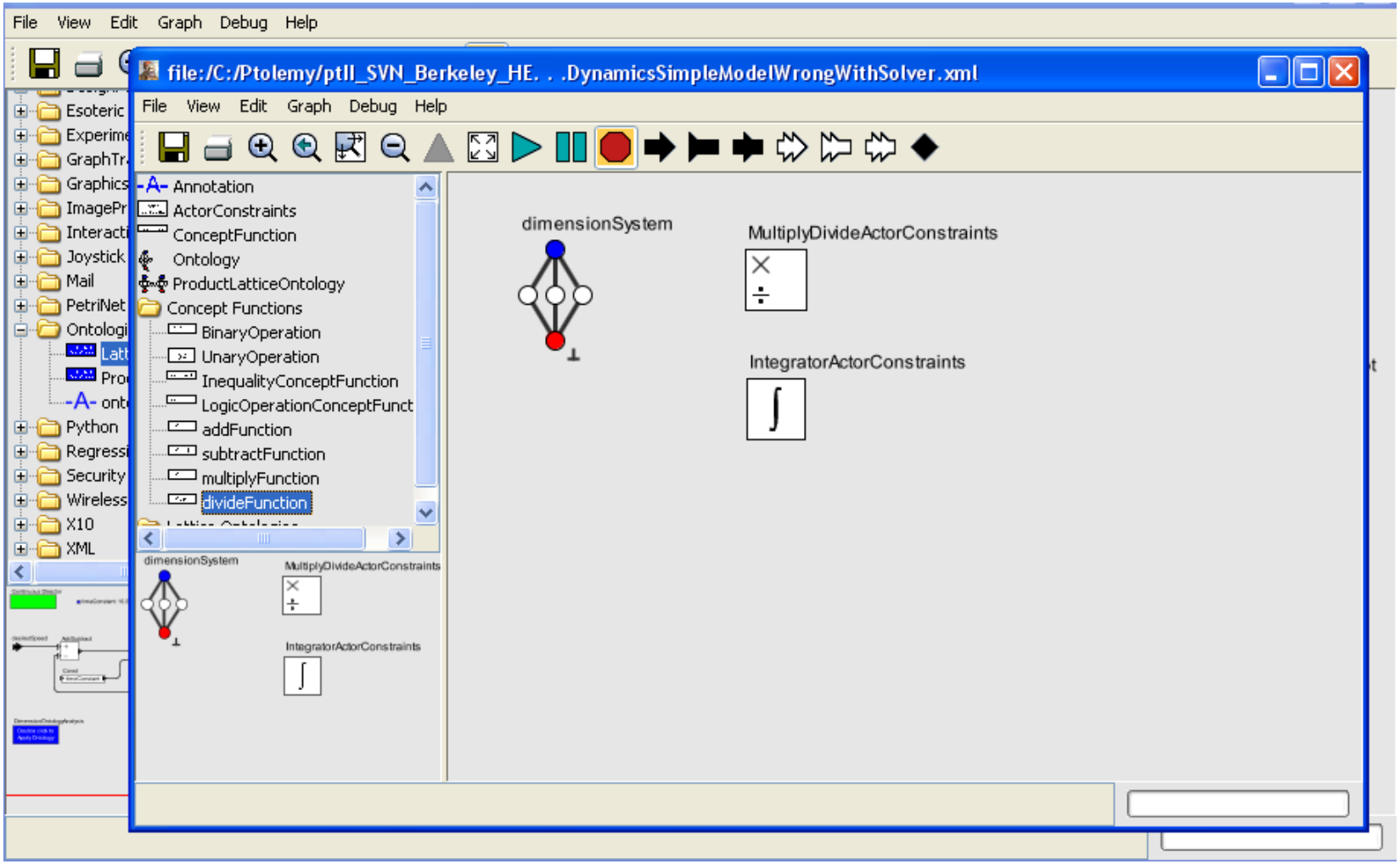
Step 5: Add Actor Constraints to the Solver Model



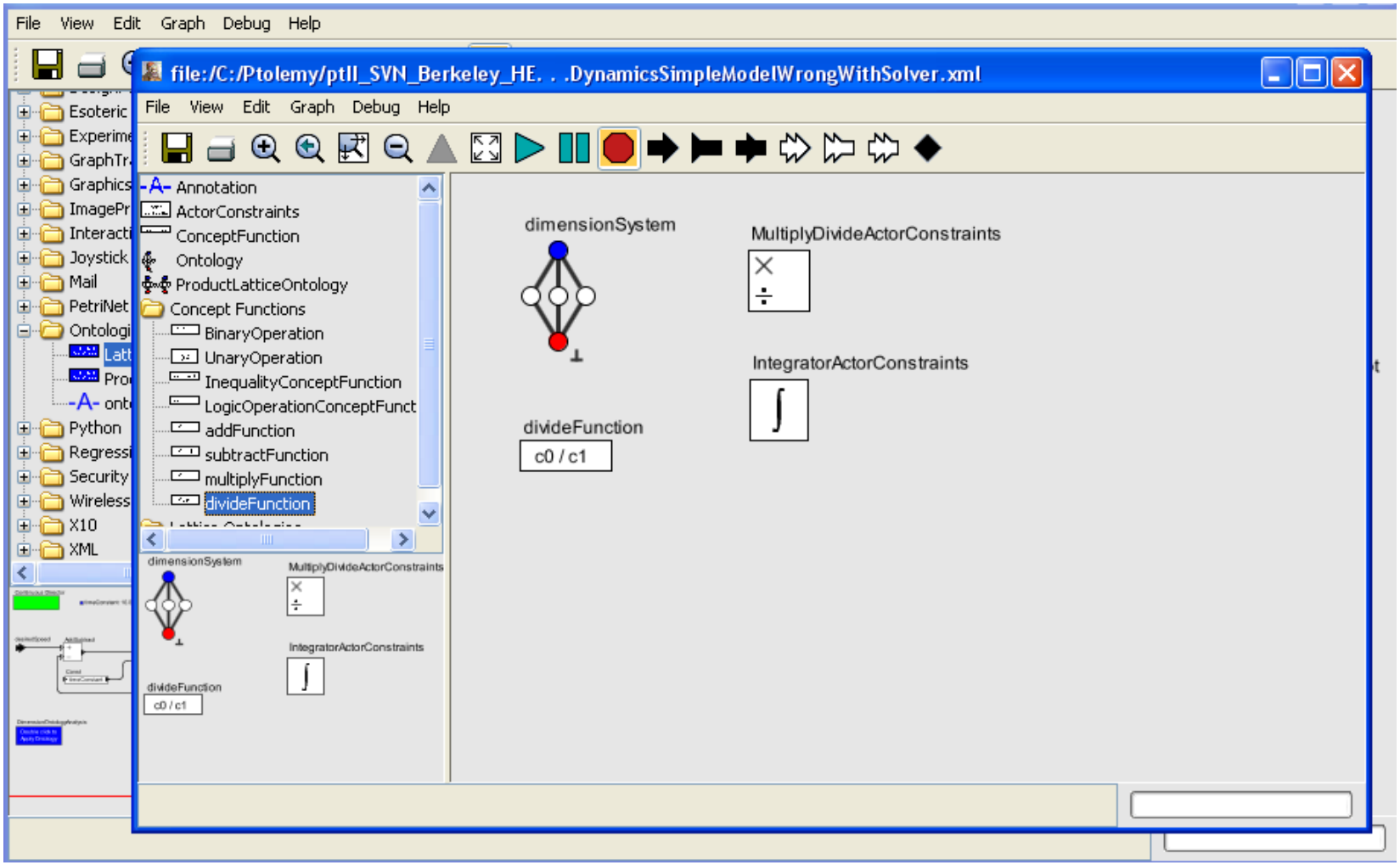
Step 5: Add Actor Constraints to the Solver Model



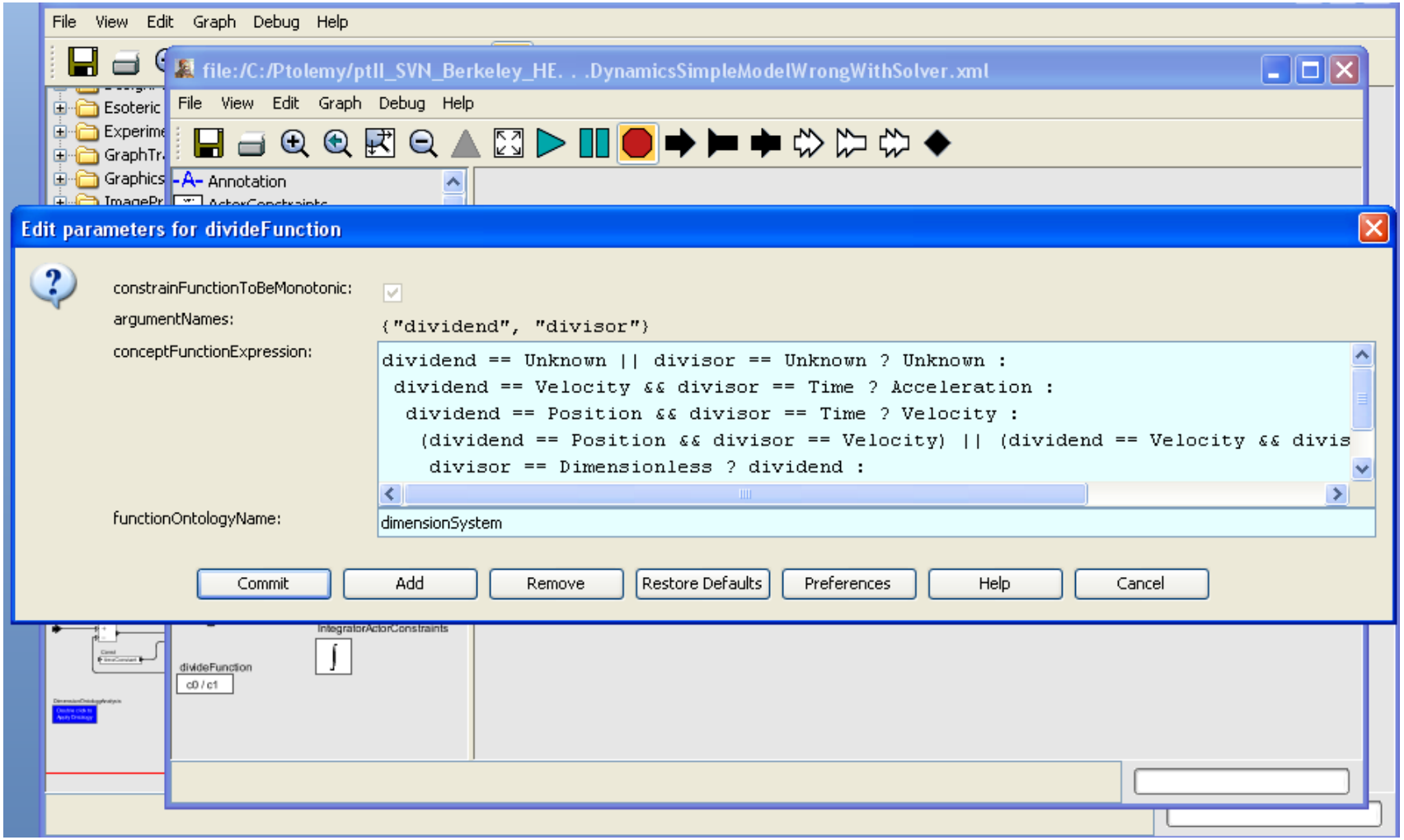
Step 5: Add Actor Constraints to the Solver Model



Step 5: Add Actor Constraints to the Solver Model

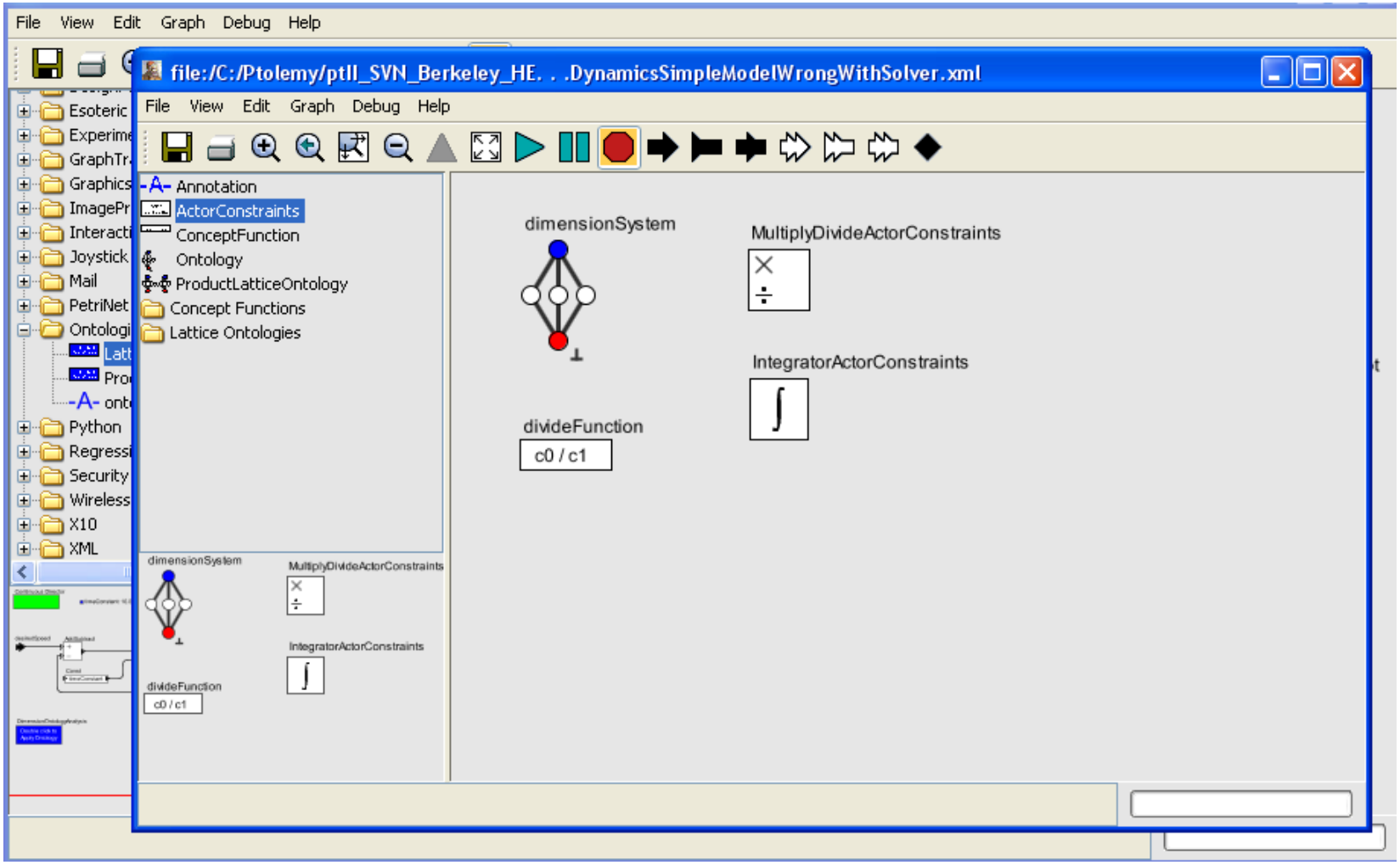


Step 5: Add Actor Constraints to the Solver Model

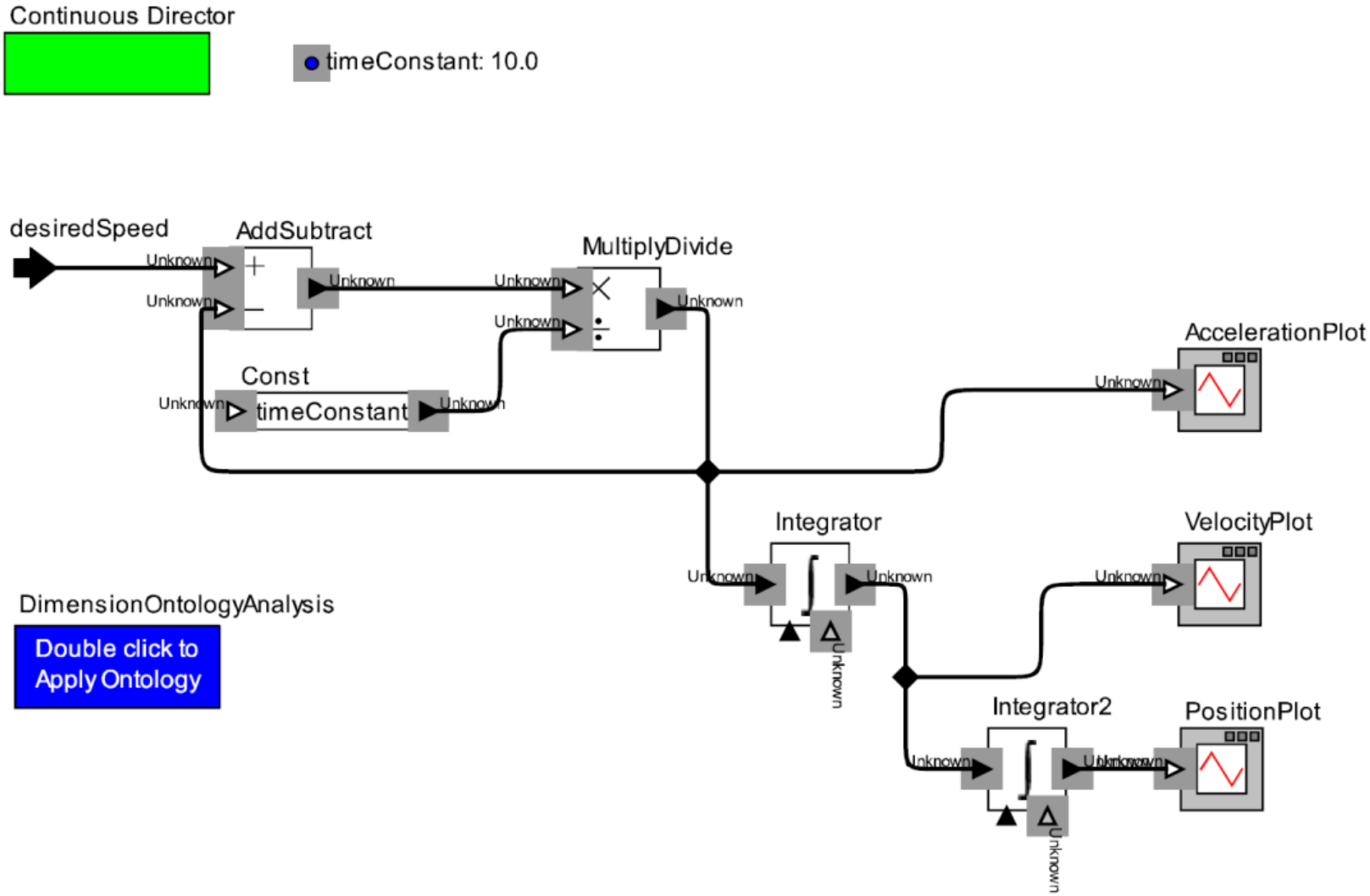


Charles Shelton, Elizabeth Latronico (Bosch), Ben Lickly, Edward Lee (UC Berkeley) | 2/16/2011 |

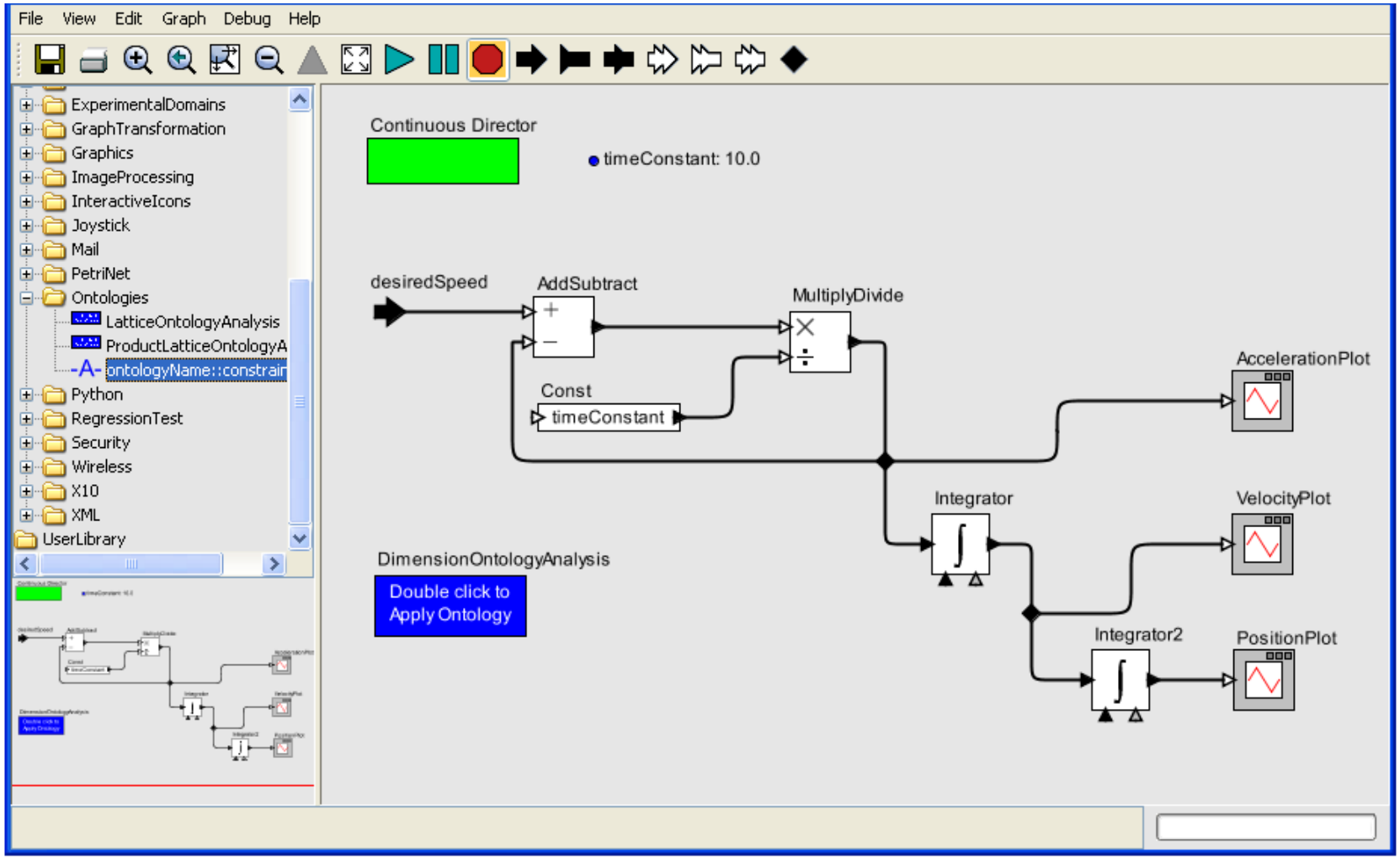
Step 5: Add Actor Constraints to the Solver Model



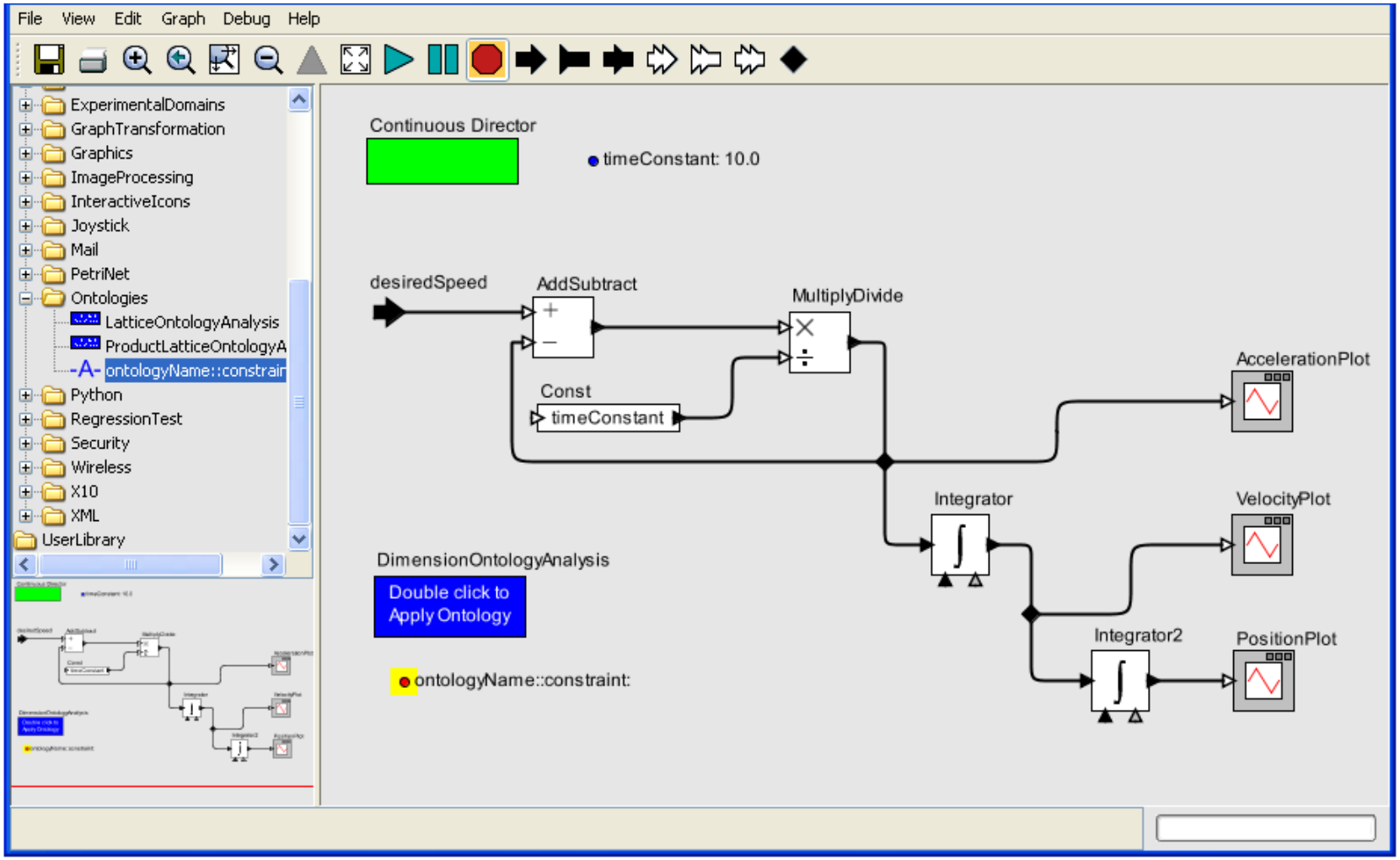
Execute the Lattice Ontology Analysis



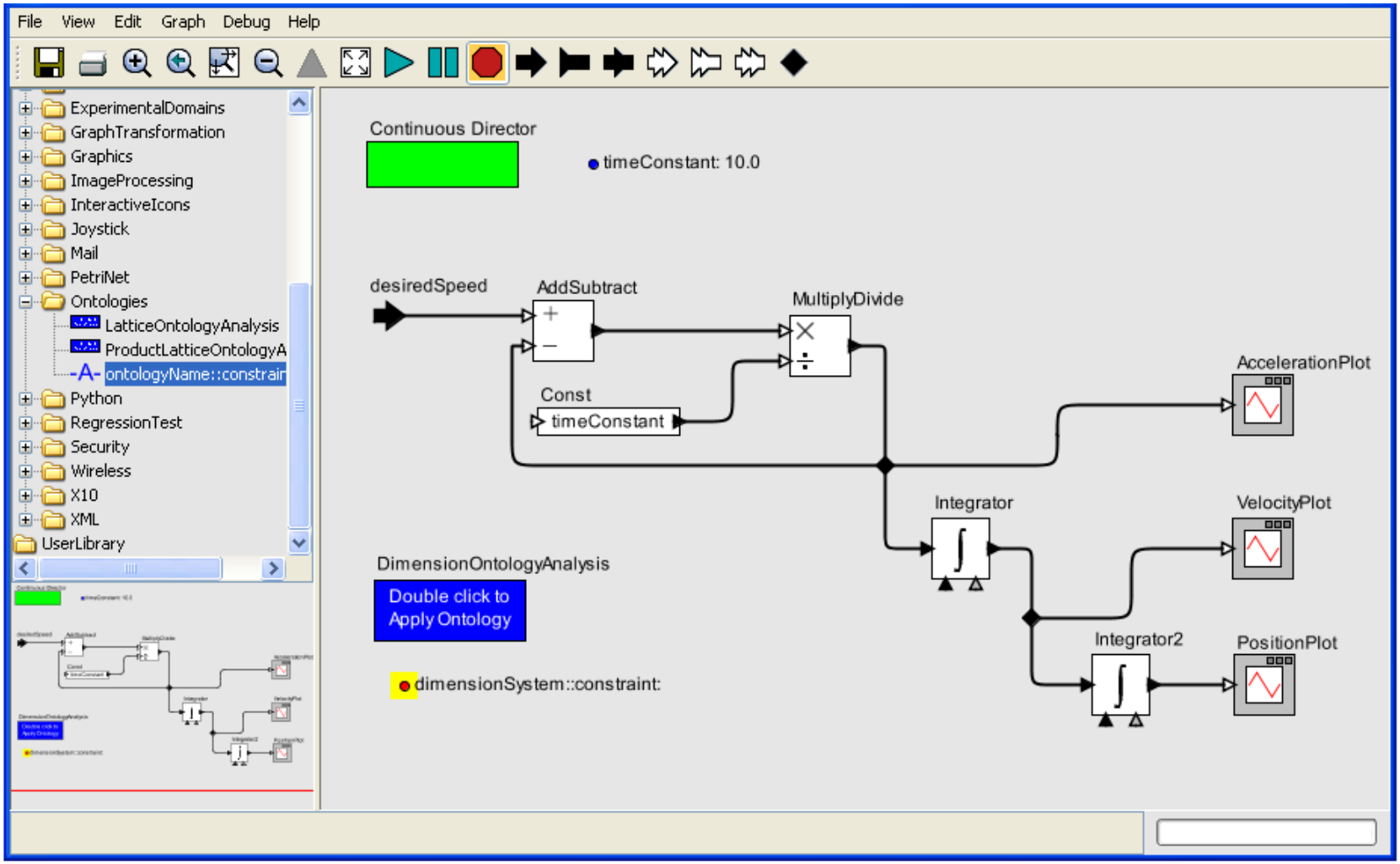
Step 6: Add Initial Constraints to the Model



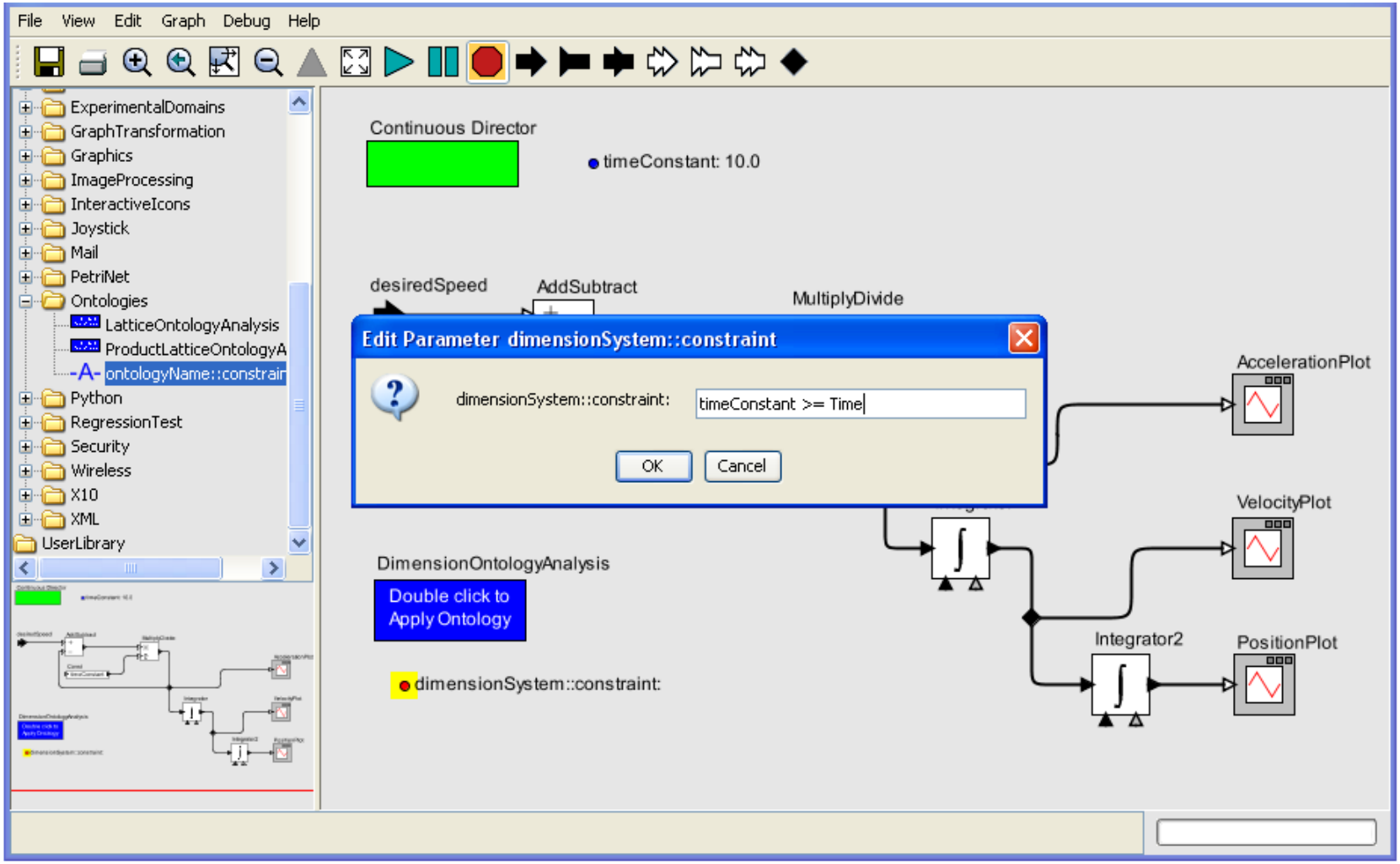
Step 6: Add Initial Constraints to the Model



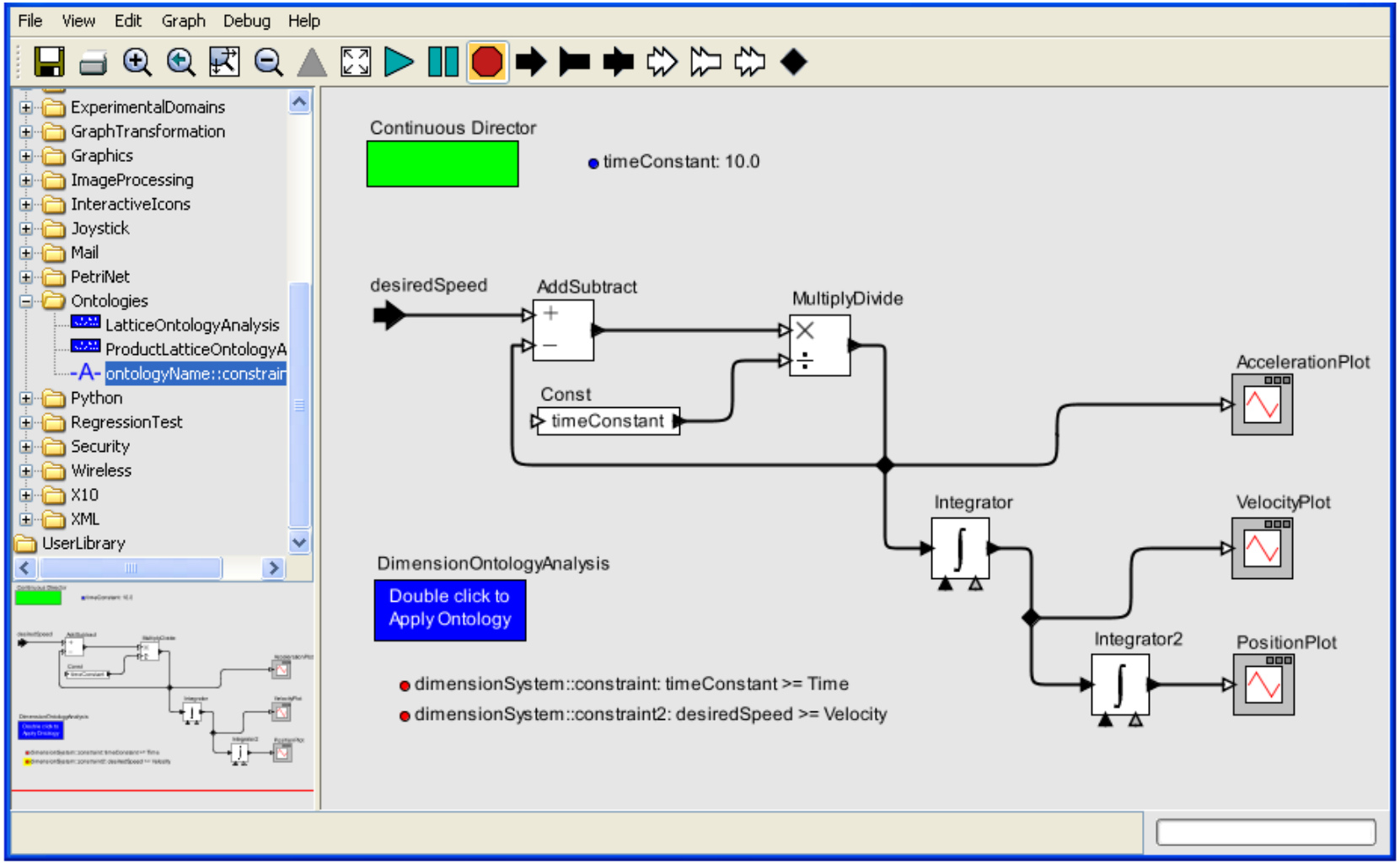
Step 6: Add Initial Constraints to the Model



Step 6: Add Initial Constraints to the Model

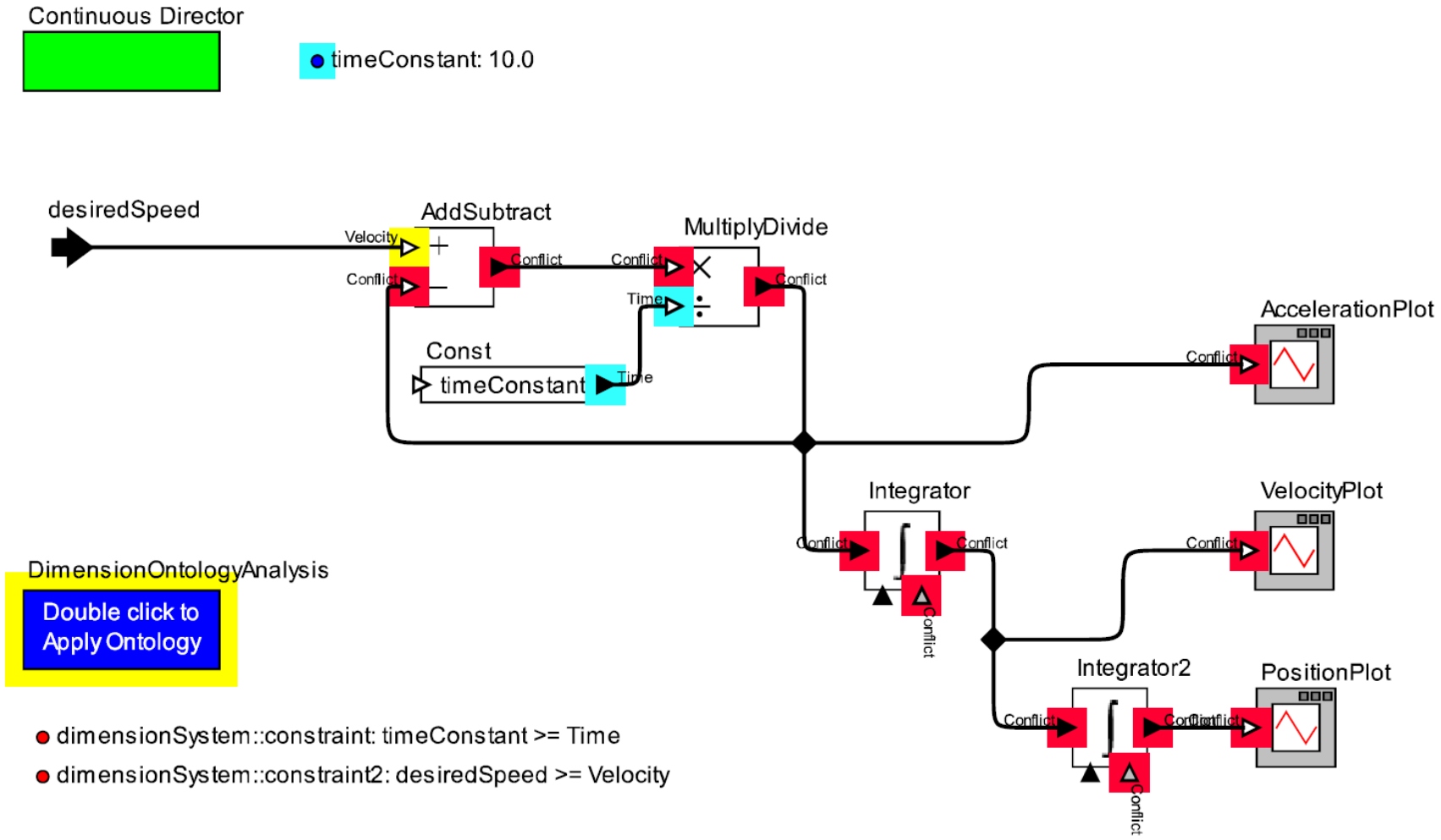


Step 6: Add Initial Constraints to the Model



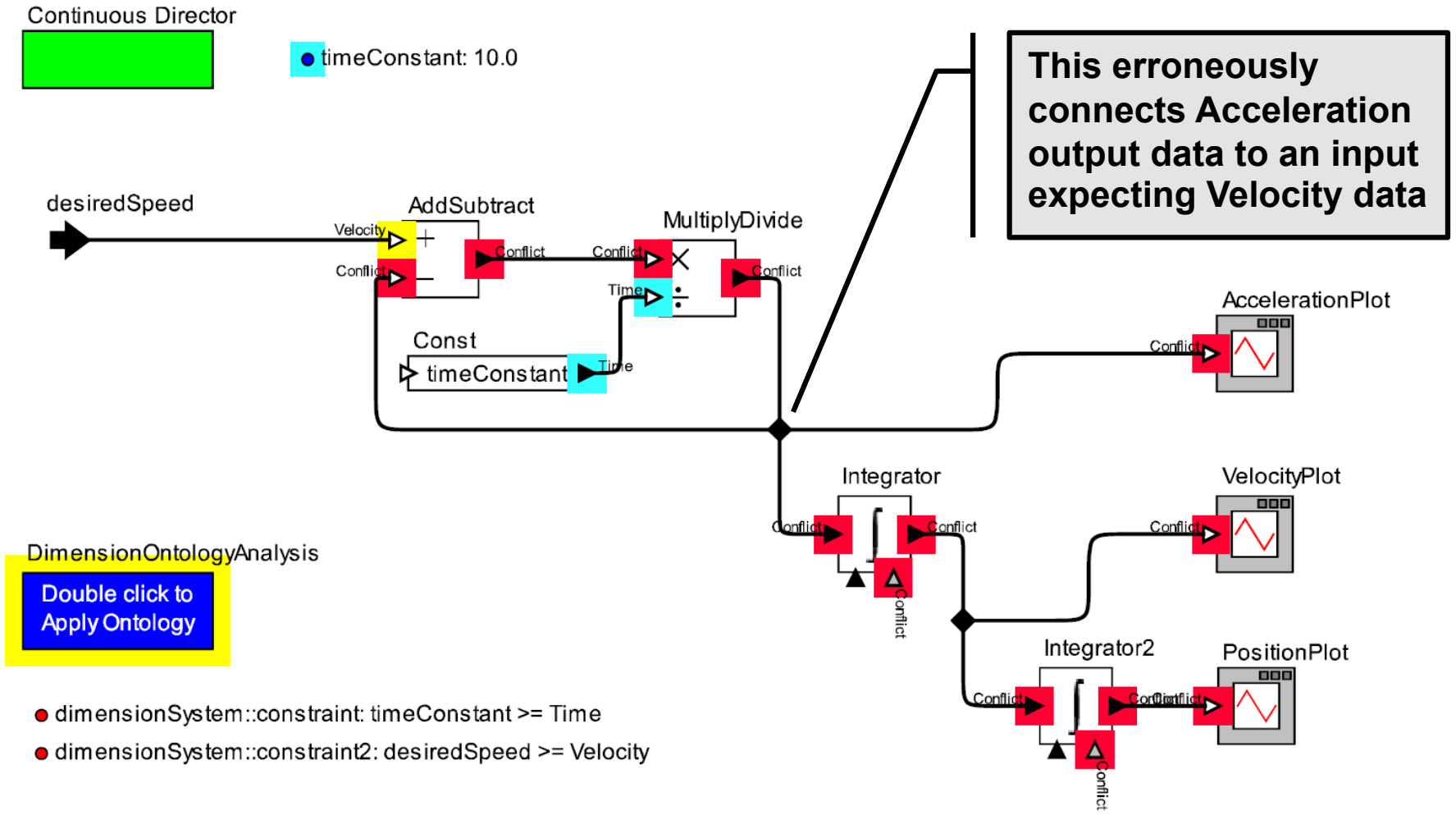
Charles Shelton, Elizabeth Latronico (Bosch), Ben Lickly, Edward Lee (UC Berkeley) | 2/16/2011 |

Reexecute the Lattice Ontology Analysis

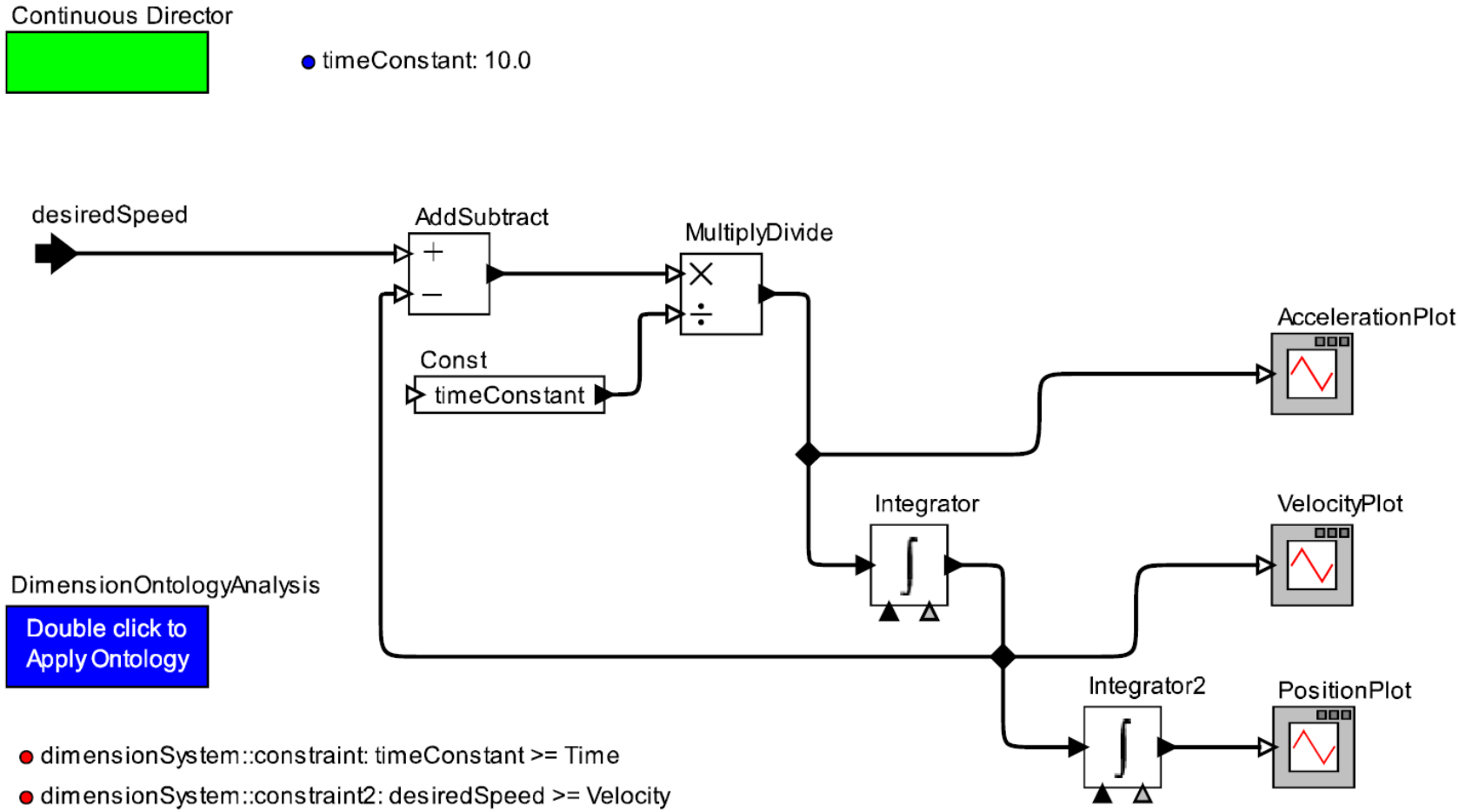


Charles Shelton, Elizabeth Latronico (Bosch), Ben Lickly, Edward Lee (UC Berkeley) | 2/16/2011 |

Fix the Model Error and Reanalyze

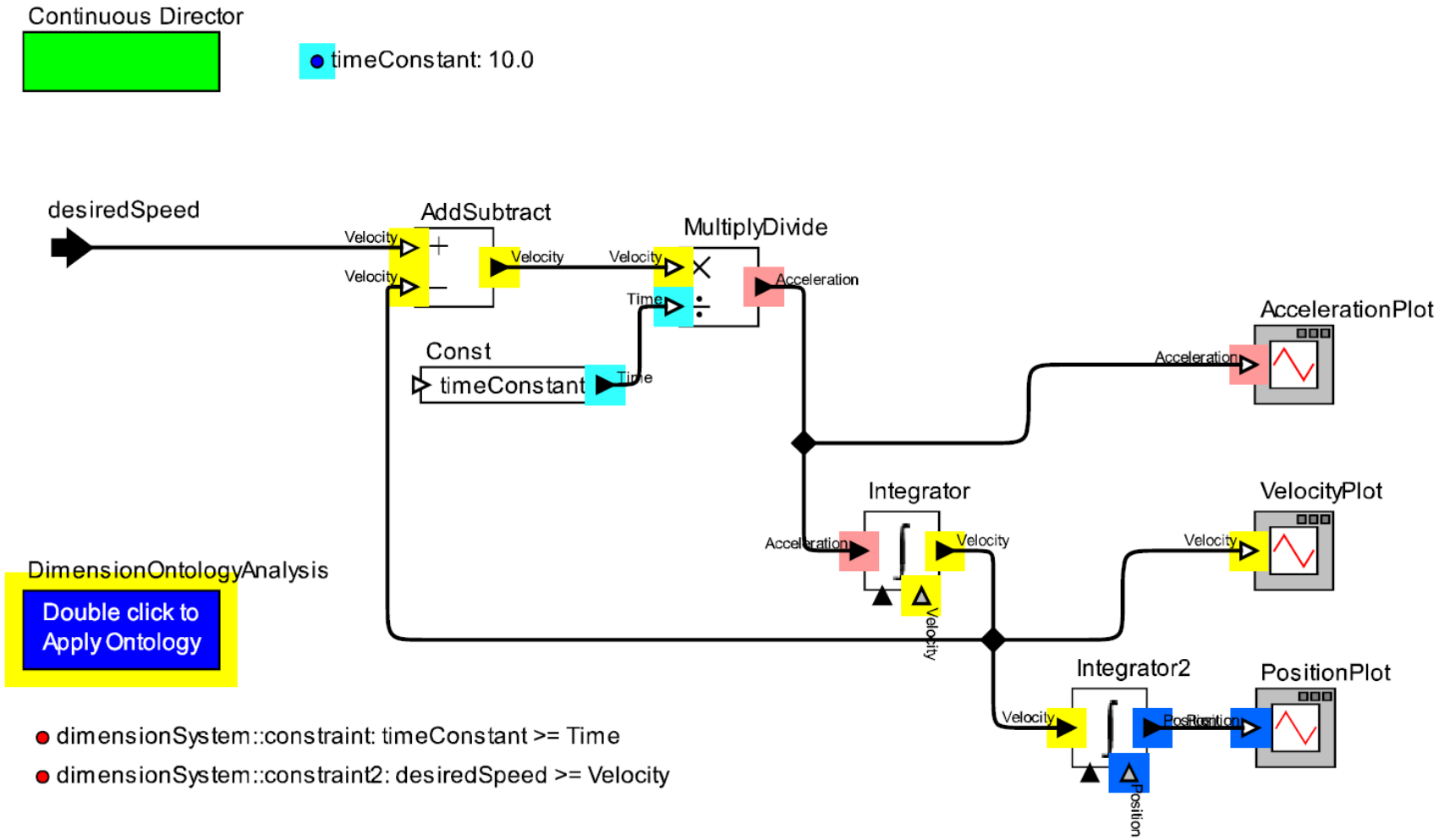


Fix the Model Error and Reanalyze



Charles Shelton, Elizabeth Latronico (Bosch), Ben Lickly, Edward Lee (UC Berkeley) | 2/16/2011 |

Done!



Potential Uses for Ontology-Based Analyses

- Type/Semantics Checking
  - Signal Data type Propagation
  - Signal Physical Dimension Propagation
  - Signal Physical/Logical Propagation
  - Signal Data/Control Propagation
- Constant/Non-Constant Propagation
- Reachability
- Observability
- Identify and Propagate Diagnostic/Functional Model Elements

Ongoing Work

- Combining multiple ontology frameworks for integrated analyses
  - **POSTER: Elizabeth Latronico**
- Ontologies with Infinite Lattice Elements
  - Constant value propagation
  - Representing and propagating records of lattice elements
  - **POSTER: Ben Lickly**
- Concept function monotonicity analysis
  - Automatically determine whether or not a function is monotonic
  - Enable easier development of ontology frameworks
- Ontology Error Analysis
  - Identify errors in the model by finding specific constraint conflicts

Conclusions

- Lattice-based ontologies enable automatic static analysis
  - Models can be verified for structural and semantic properties
  - Guaranteed sound analysis given:
    - The ontology is a lattice
    - All constraint functions are monotonic
  - Analysis algorithm scales with the number of constraints
    - # constraints scales with # model elements

- Ontologies Package Demos in the Ptolemy Repository
  - `/ptolemy/data/ontologies/demo`
  - `/ptolemy/data/ontologies/demo/DimensionSystemExample`
  - `/ptolemy/data/ontologies/demo/CarTracking`

- Thanks!
  - Charles.Shelton@us.bosch.com