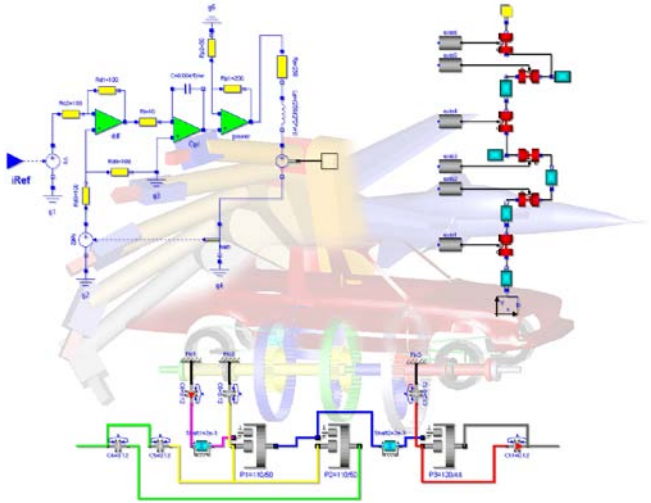
Integrated Model-Based Development with OpenModelica and ModelicaML

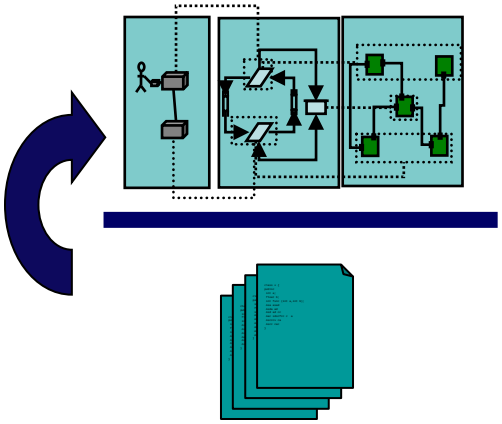
partly in the OPENPROD project

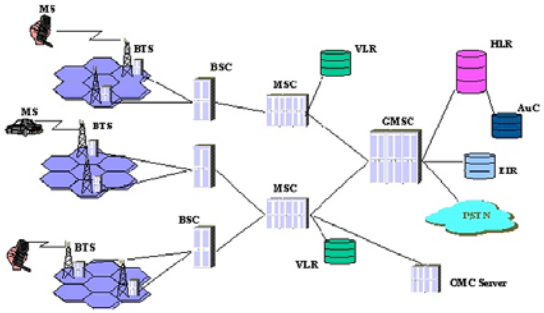
Linköping University

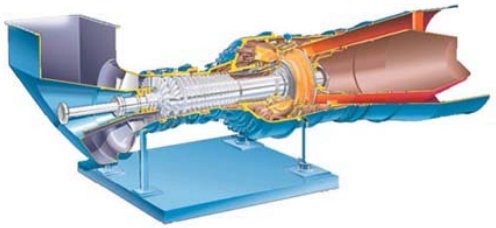
2010-02-10

Peter Fritzson





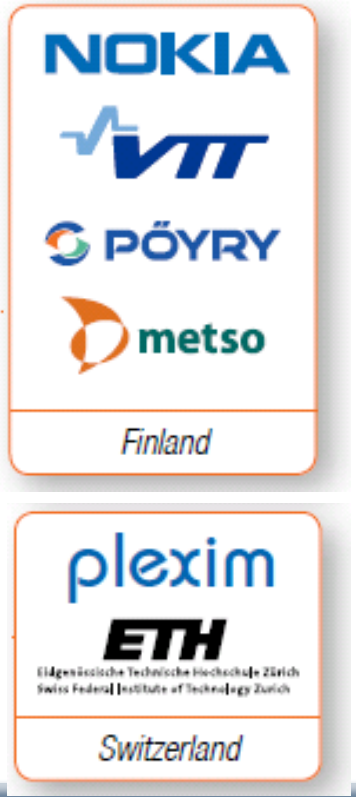




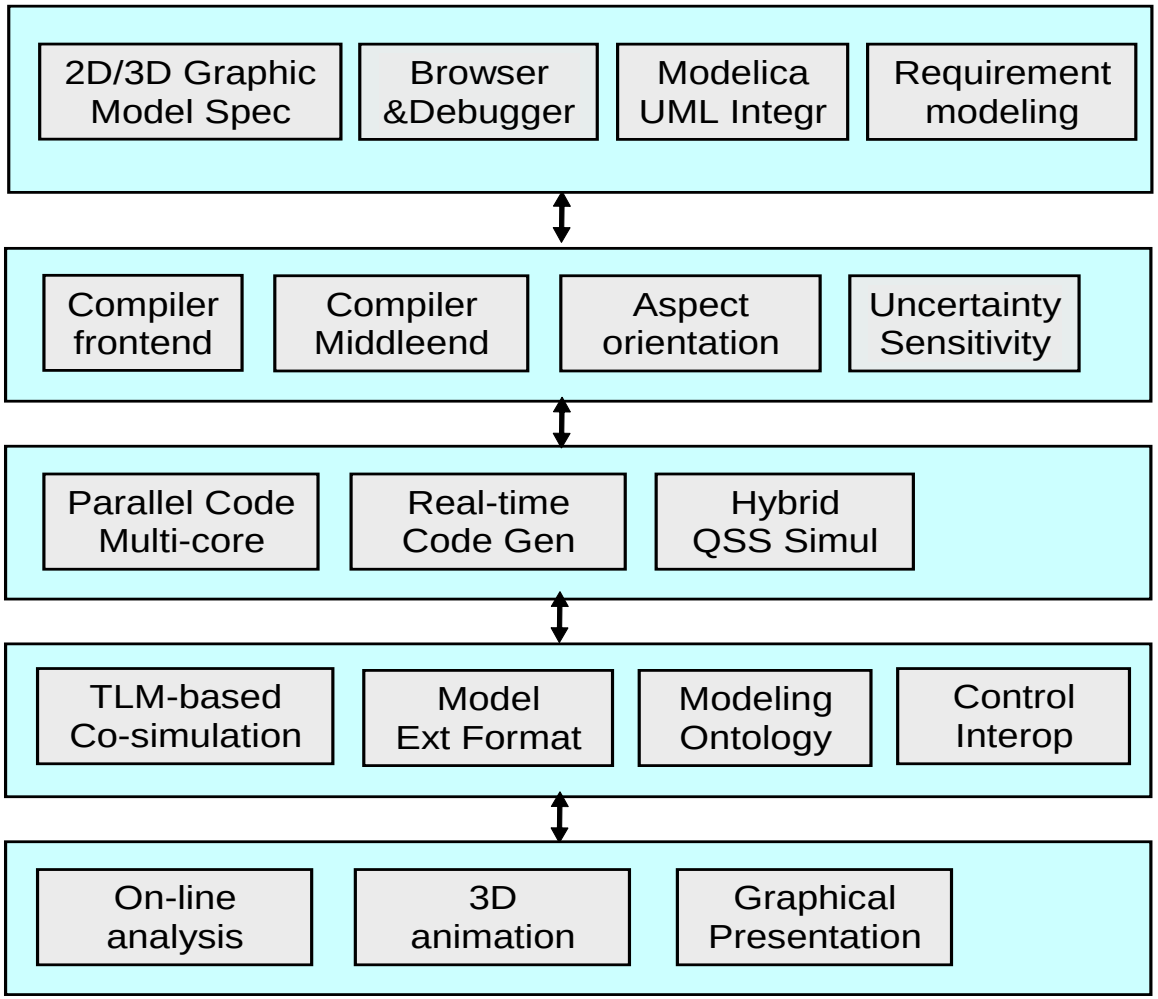
MODPROD Center Associated Project: OPENPROD – Large 3-year ITEA2 EU Project

28 partners from 5 countries: 11 large industries,
7 SMEs, 5 research institutes, and 5 universities.
Project size: > 90 person years,
Budget: About 11 Mill. €, Duration June 2009 - Aug 2012.

Coordination by Sune Horkeby, Peter Fritzson



OPENPROD System Structure



Model-Driven Development Environment, WP2

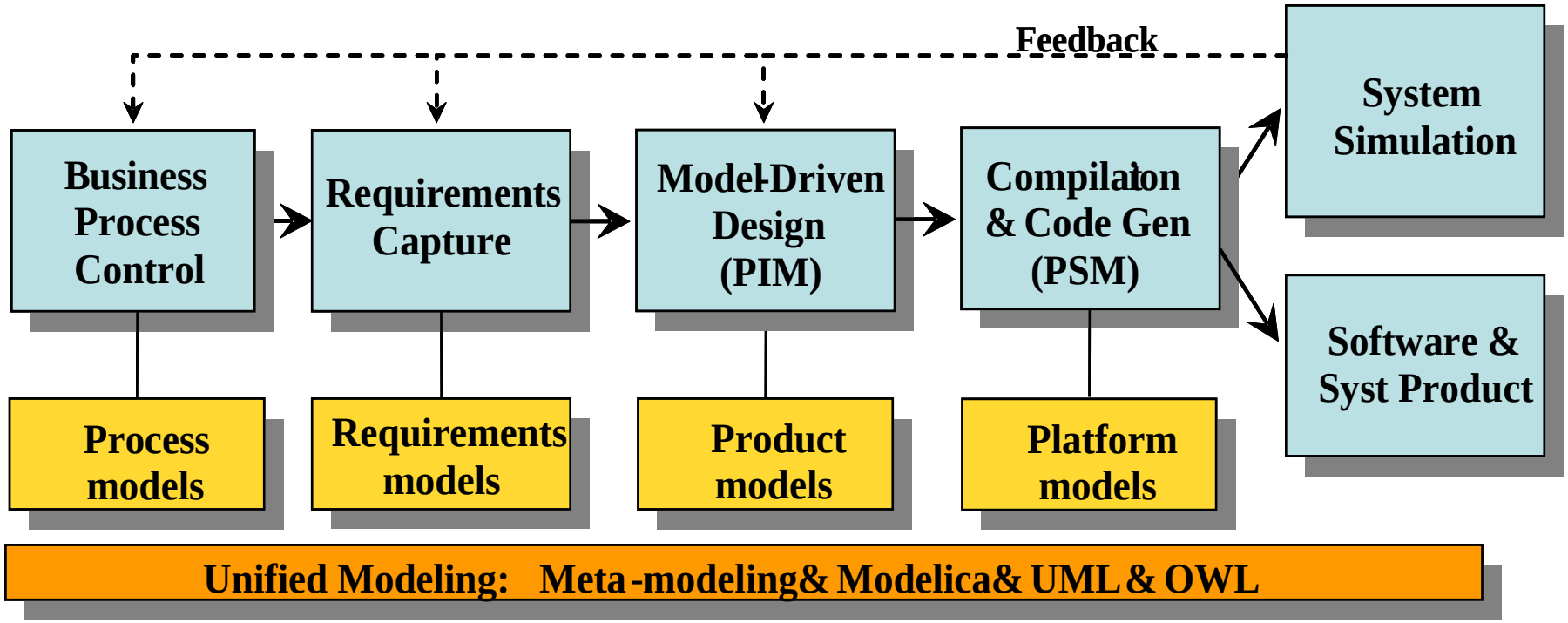
Model Compiler WP3

CodeGen & Simulation WP4

Interoperability WP5

Graphical Presentation WP2

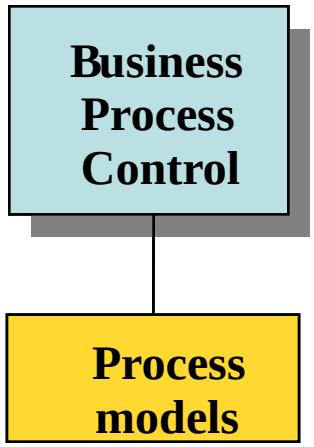
OPENPROD Vision of Integrated Model-Based Product Development



OPENPROD Vision of unified modeling framework for model-driven product development from platform independent models (PIM) to platform specific models (PSM)

Current work based on Eclipse, UML/SysML, OpenModelica

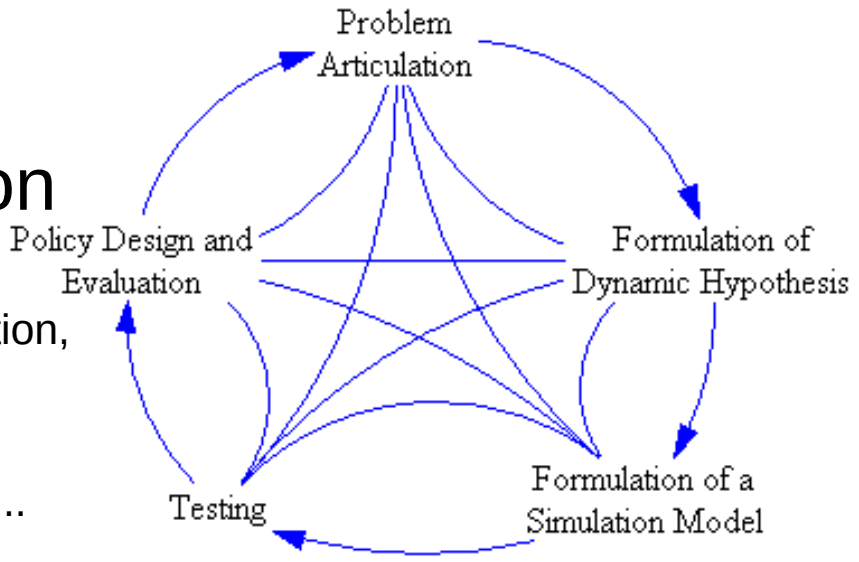
Business Process Models



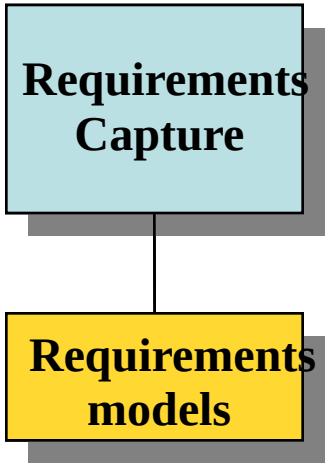
- In OPENPROD WP2:
- VTT develops business processing tool using Simantics and OpenModelica, based on System Dynamics.
- Industrial applications by partners: Nokia, Metso, Pyöru

Modeling Business Processes (VTT)

- Problem Articulation
  - Problem, variables, time scale, interfaces…
  - (Model only aspects relevant to the problem)
- Formulation of Dynamic Hypothesis
  - Hypothesis, causal relations
- Formulation of a Simulation Model
  - Structure, submodels, parameter estimation, ...
- Testing
  - Comparison to historical data, sensitivity...
- Policy Design and Evaluation
  - Scenarios, new policies, strategies, ”what if”-simulation, sensitivity in different situations

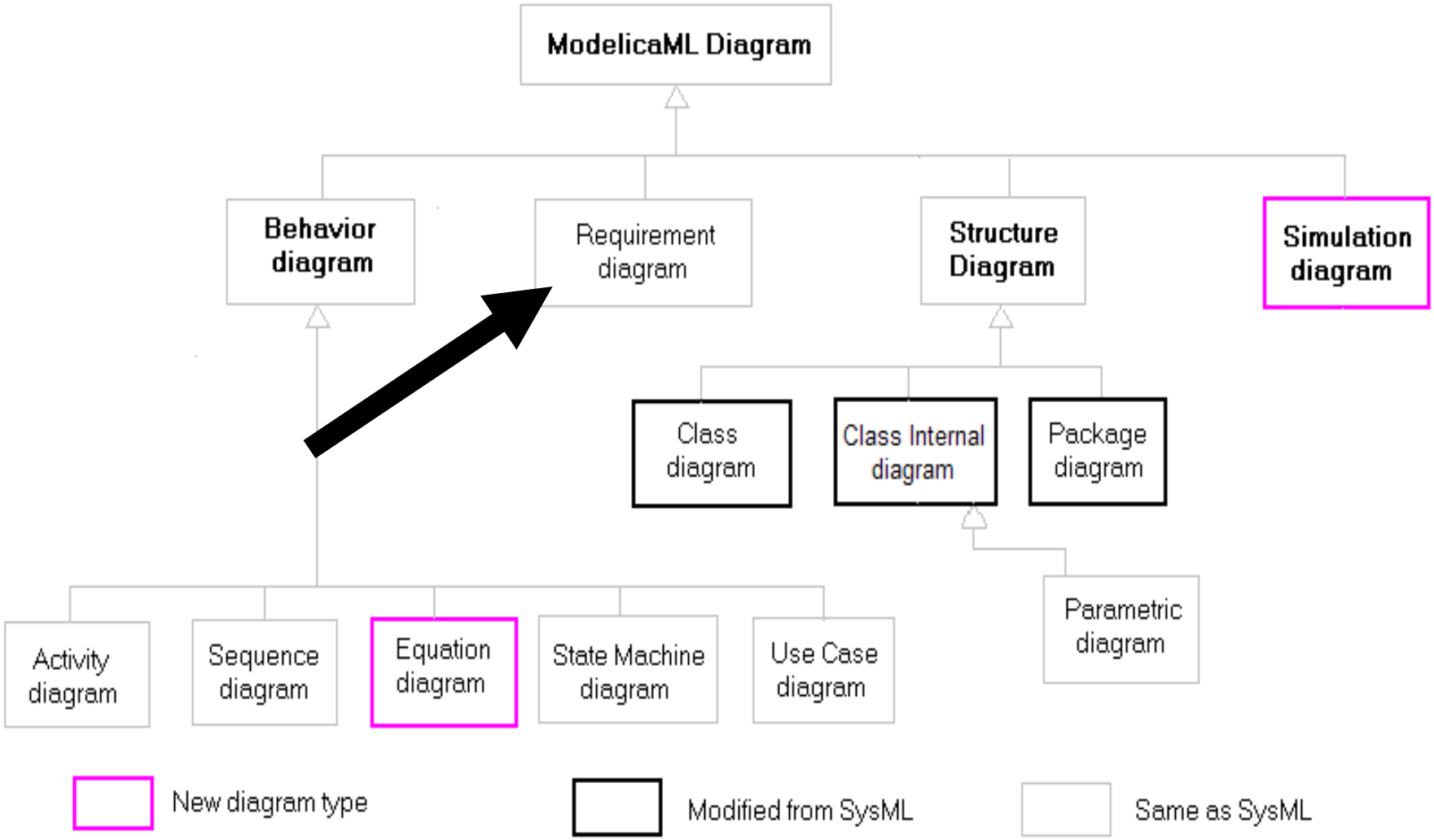


Capturing and Transformations of Requirements

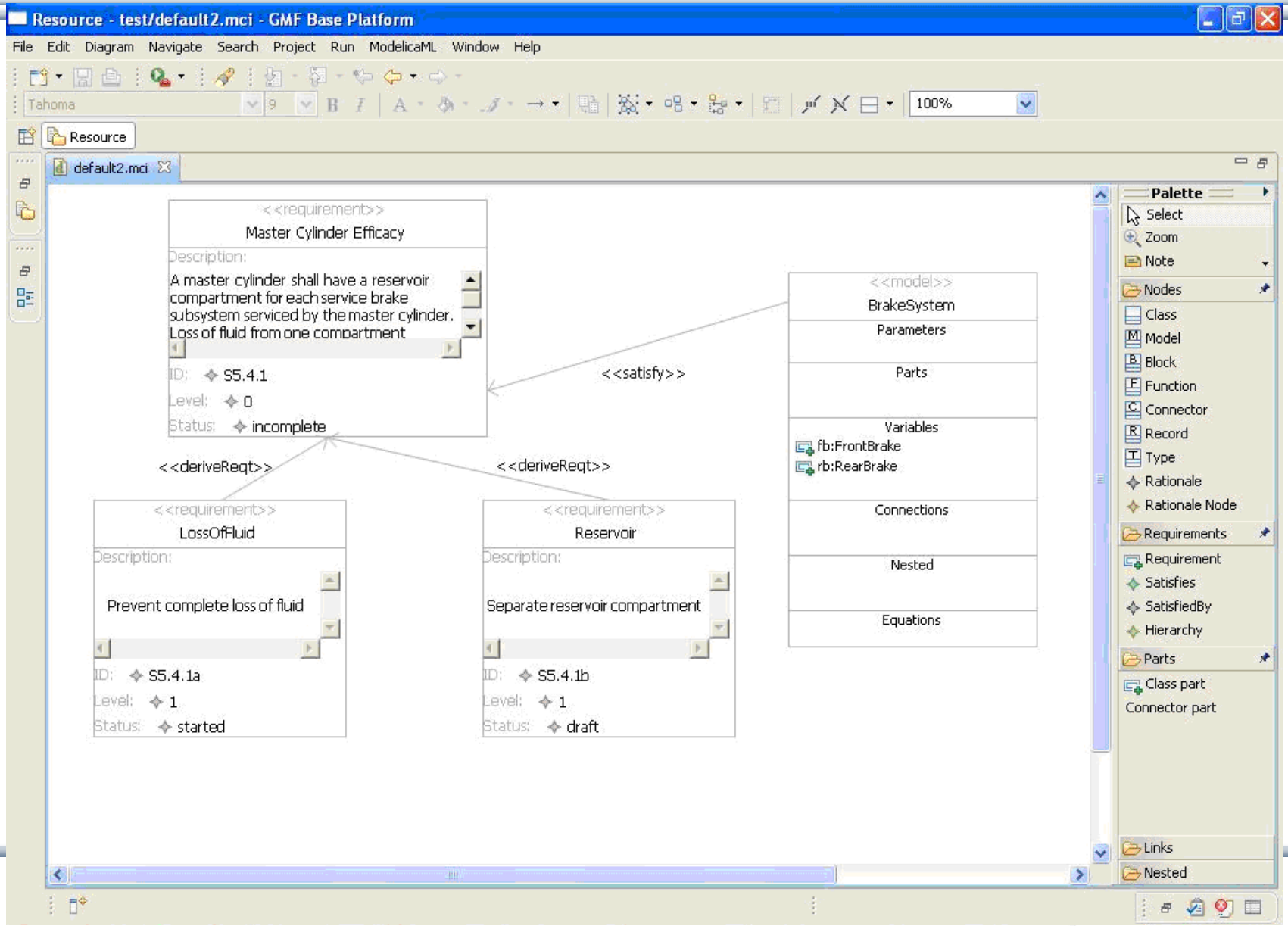


- Approach 1: SysMLText boxes
- Approach 2: ModelicaML – requirements in text and slightly formalized
- Approach 3: Use the behavior engineering approach on capturing and formalizing requirements

Approach 1: SysML Requirement diagrams (= text boxes) in ModelicaML-2007

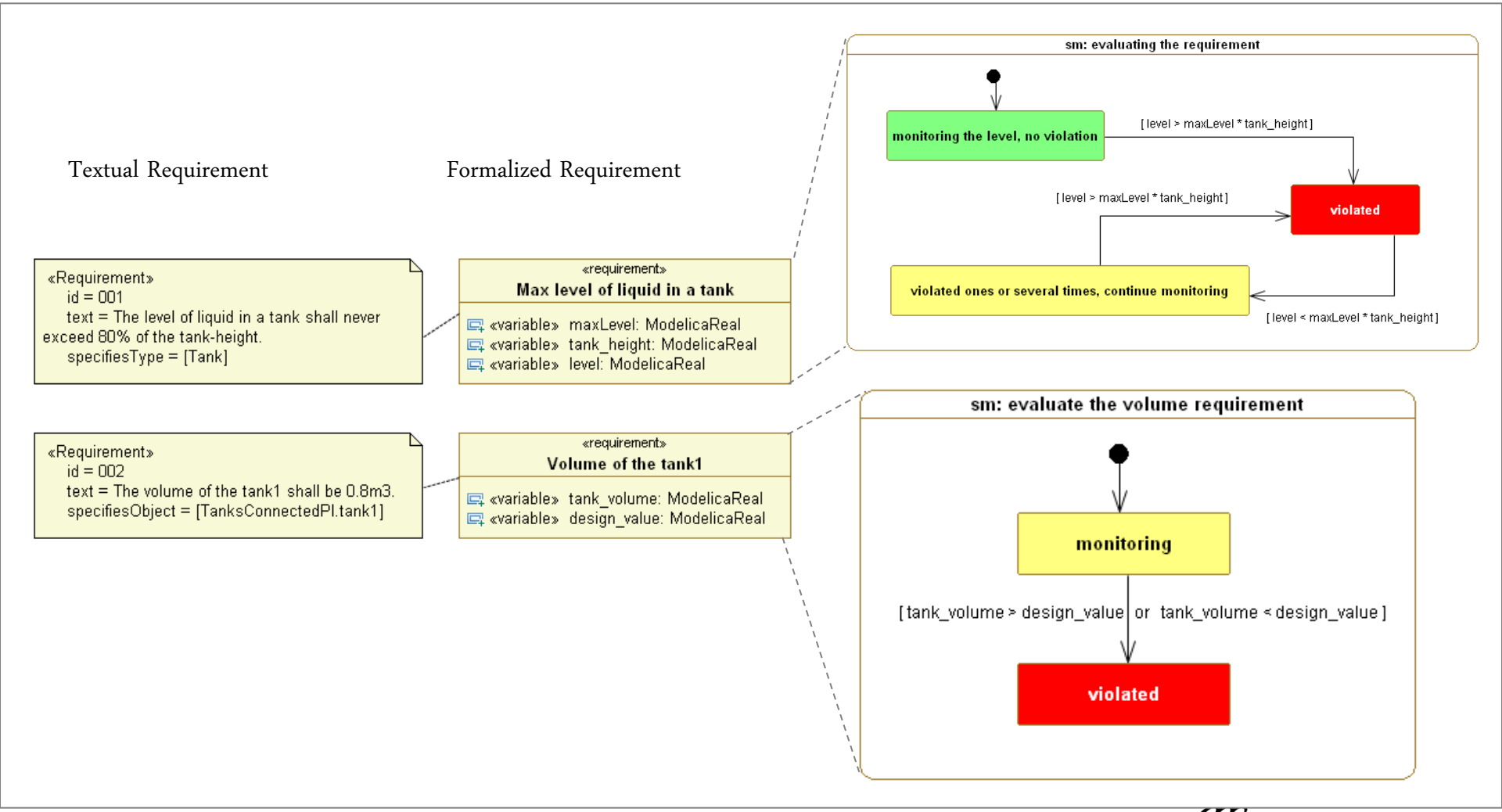


Approach 1 &2: Requirements Modeling in Eclipse using ModelicaML-2007. Also use equations



9

Approach 1 & 2: ModelicaML-2009
Example: Representation of System Requirements



ModelicaML-2009: Simulation and Requirements Evaluation

«model»
(TwoTanksSystemExample::SystemSimulations)
TankSystemSimulation

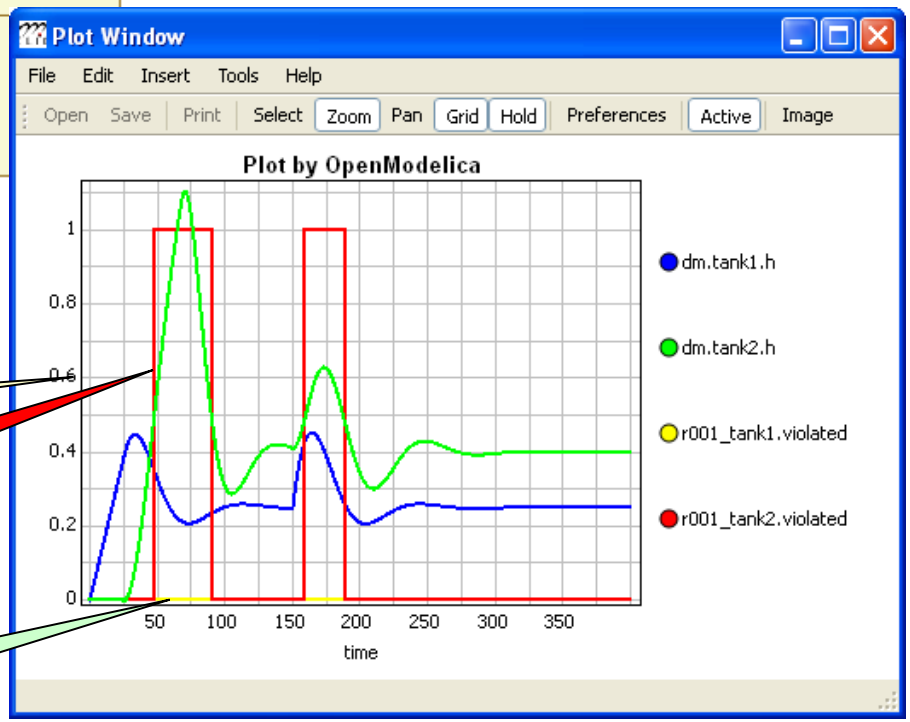
«component» dm: TanksConnectedPI
«requirementInstance» r001_tank1: Max level of liquid in a tank
«requirementInstance» r001_tank2: Max level of liquid in a tank
«requirementInstance» r002_tank1: Volume of the tank1

Req. 001 is instantiated 2 times (there are 2 tanks in the system)

tank-height is 0.6m

Req. 001 for the tank2 is violated

Req. 001 for the tank1 is not violated



Approach 3 – Use Behavior Engineering for More Formal Requirements Capture and Analysis

Developed by Prof. Geoff Dromey, Griffith Univ, Brisbane

- 5 Large-scale industry projects
  - In Defence, Transportation, Banking and Finance
  - Between 800-1250 requirements
- All previously reviewed with respective organisations internal review processes
- Defect detection rate approximately 2 to 3 times that of traditional ad-hoc, checklist-based, and scenario-based reading techniques reported in Porter, 1998.

Requirements Evaluation Using Behavior Trees

Findings from Industry

Daniel Powell

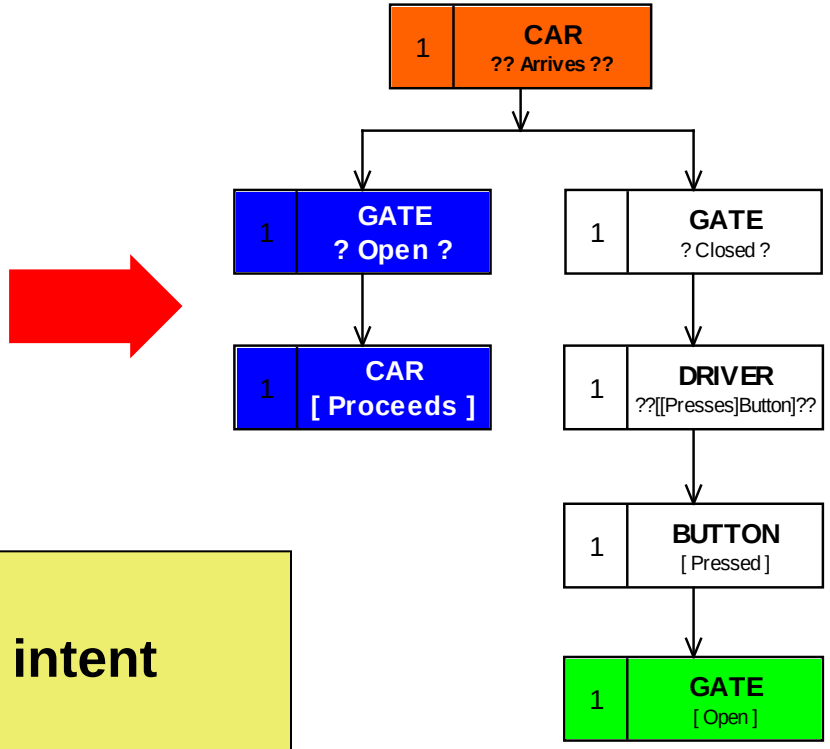
http://aswec07.cs.latrobe.edu.au/5.zip

Formalization - Requirements Translation

Functional Requirement

When a car is arrives,
if the gate is open the car proceeds,
otherwise if the gate is closed, when
the driver presses the button
it causes the gate to open

Behavior Tree



Formalization
- clarification and preservation of intent
- strict use of original vocabulary
- removes ambiguity, aliases, etc
- aids stakeholder validation, understanding
- approaches repeatability

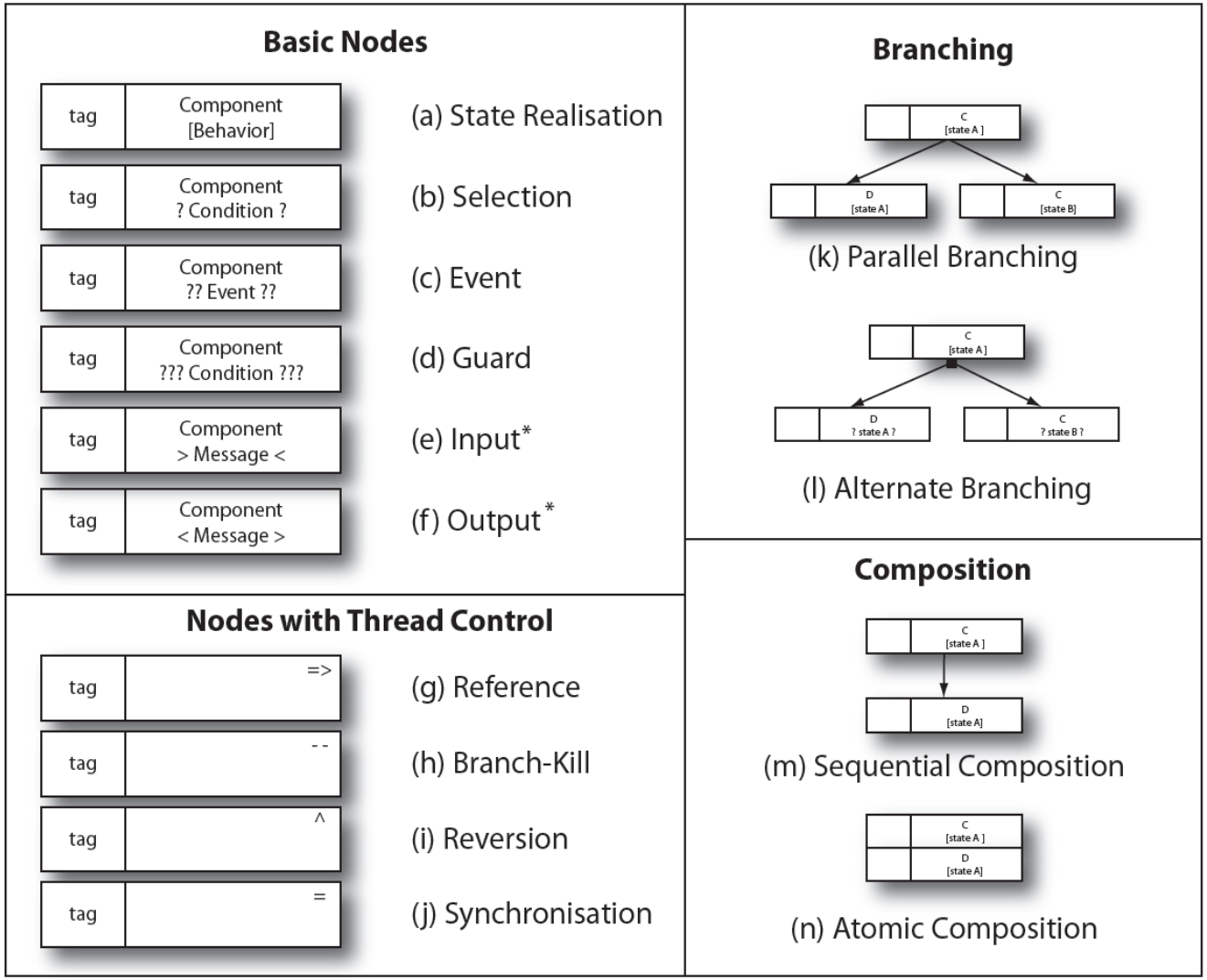
A Brief Introduction to Behavior Engineering (BE)

- Behavior Engineering (BE) acronyms …

Behavior Modeling Process (BMP)	Behavior Modeling Language (BML)	
	Behavior Trees (BT)	**Composition Trees (CT)**
Requirements Translation	Requirement Behavior Trees (RBTs)	Requirement Composition Tree (RCT)
Requirements Integration	Integrated Behavior Tree (IBT)	Integrated Composition Tree (ICT)
System Specification	Model Behavior Tree (MBT)	Model Composition Tree (MCT)
System Design	Design Behavior Tree (DBT)	Design Composition Tree (DCT)

A Brief Introduction to Behavior Engineering
Summary of Behavior Tree Notation

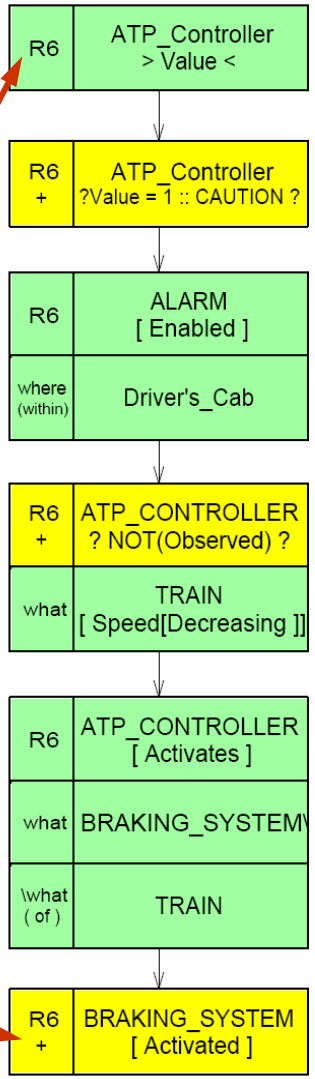
Summary of the Behavior Tree Notation



Translation from a Requirement in English to a Requirement Behavior Tree (RBT)

How to translate from a Requirement in Natural Language to an RBT

R6. If a caution signal is returned to the ATP controller then the alarm is enabled within the driver's cab. Furthermore, once the alarm has been enabled, if the speed of the train is not observed to be decreasing then the ATP controller activates the train's braking system.



The Tag traces these Behavior Tree nodes back to Requirement 6.

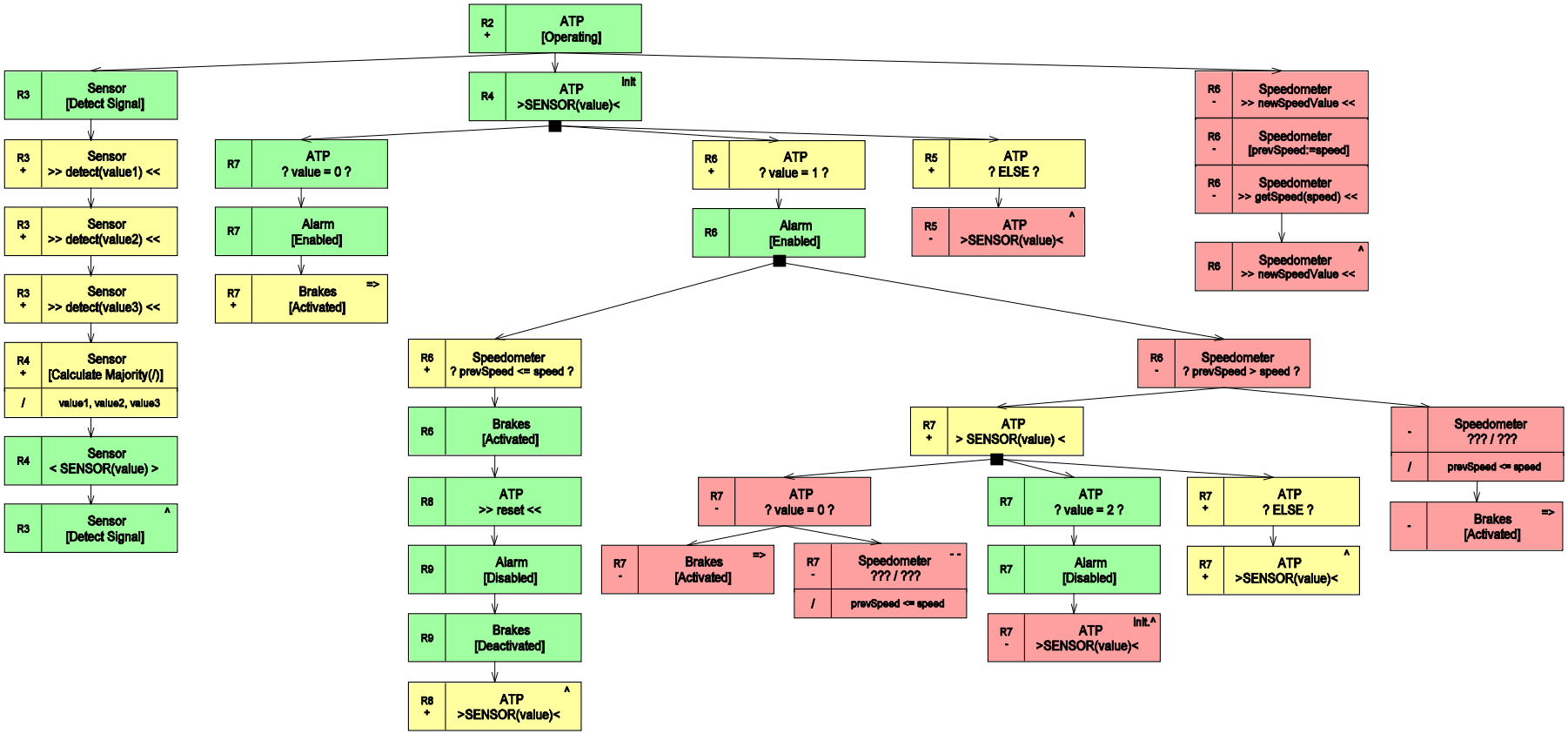
A '+' and a yellow color denote the behavior is implied by the requirements

Red color denotes behavior is missing in the requirements

Case Study: An Automated Train Protection System

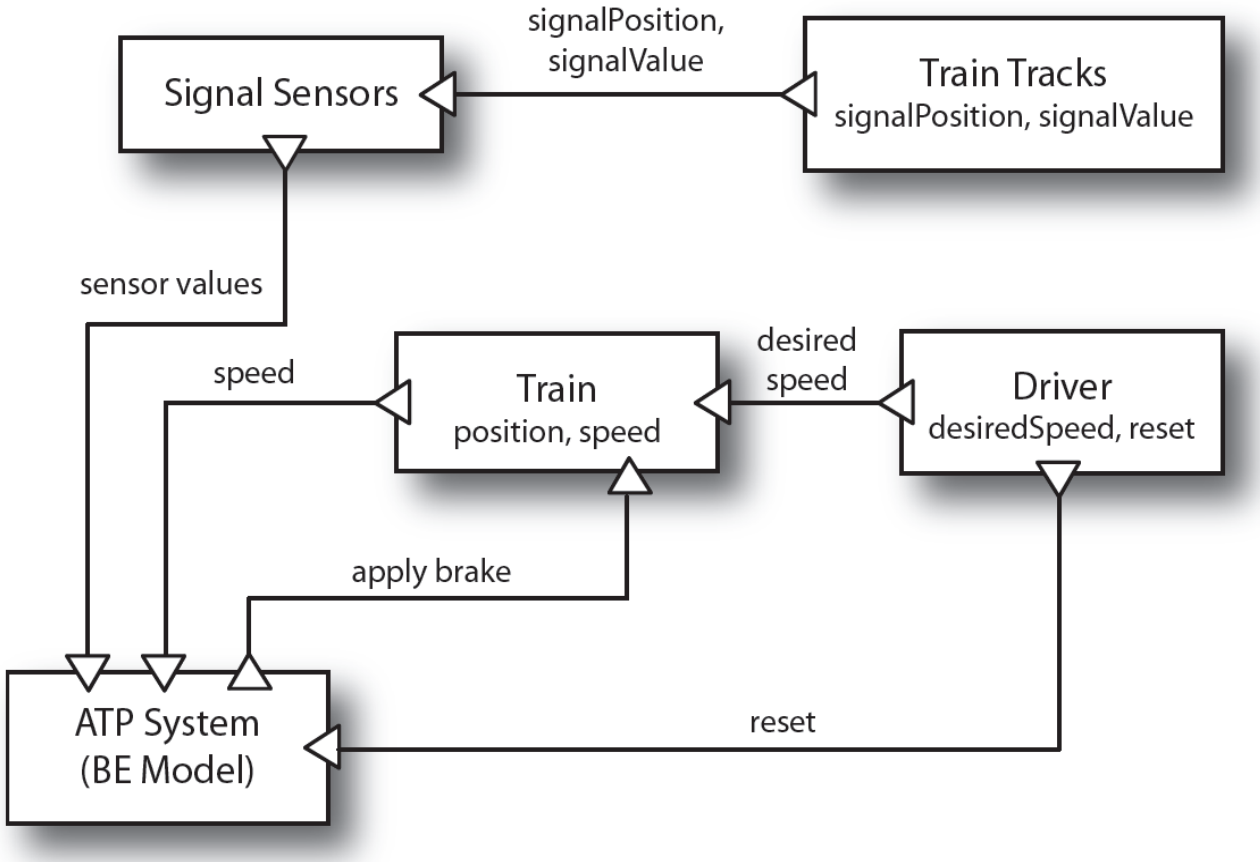
BE Model of the ATP System

(yellow: implied from requirements, red: missing)

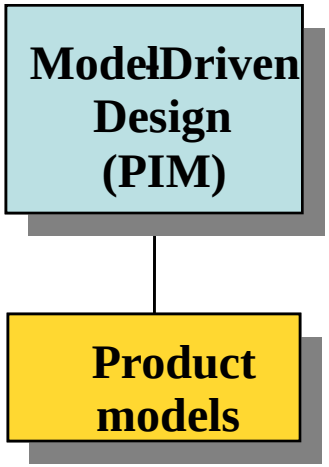


Case Study:
An Automated Train Protection System

Modelica Model of the ATP System (graphical view)

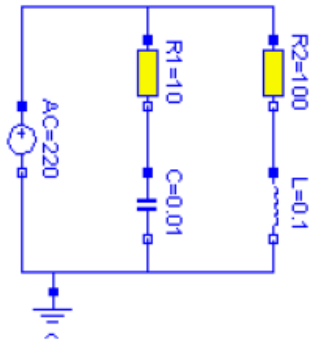


Modeldriven Design

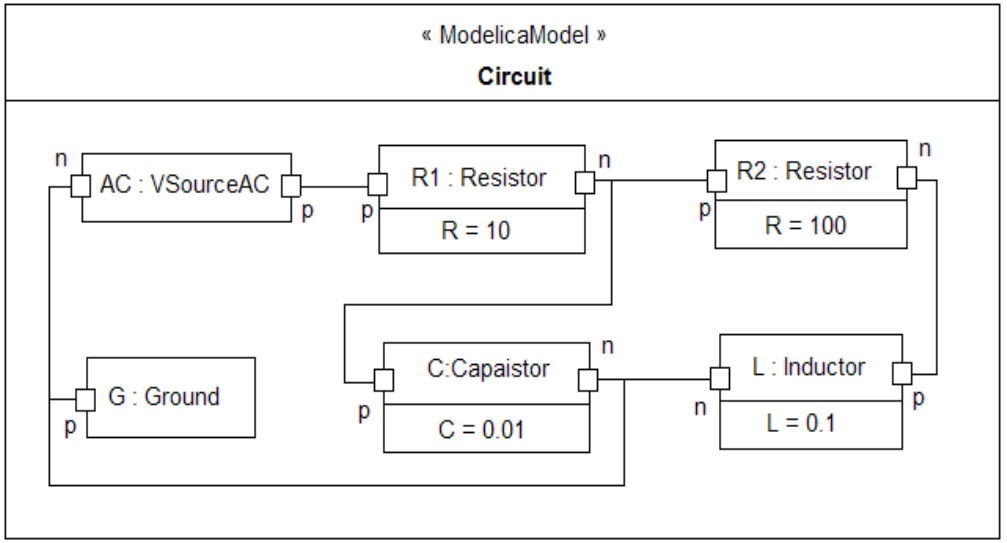


- Graphical modeling of software and systems using UML and SysML
  - software models, system overview models
- Graphical modeling using Modelica
  - Models of physical systems and embedded system software as well as system architecture
- Textual modeling using Modelica
  - OpenModelica MDT Eclipse plugin

Modelica Graphic Connection Diagram and ModelicaML/UML Class Internal Diagram

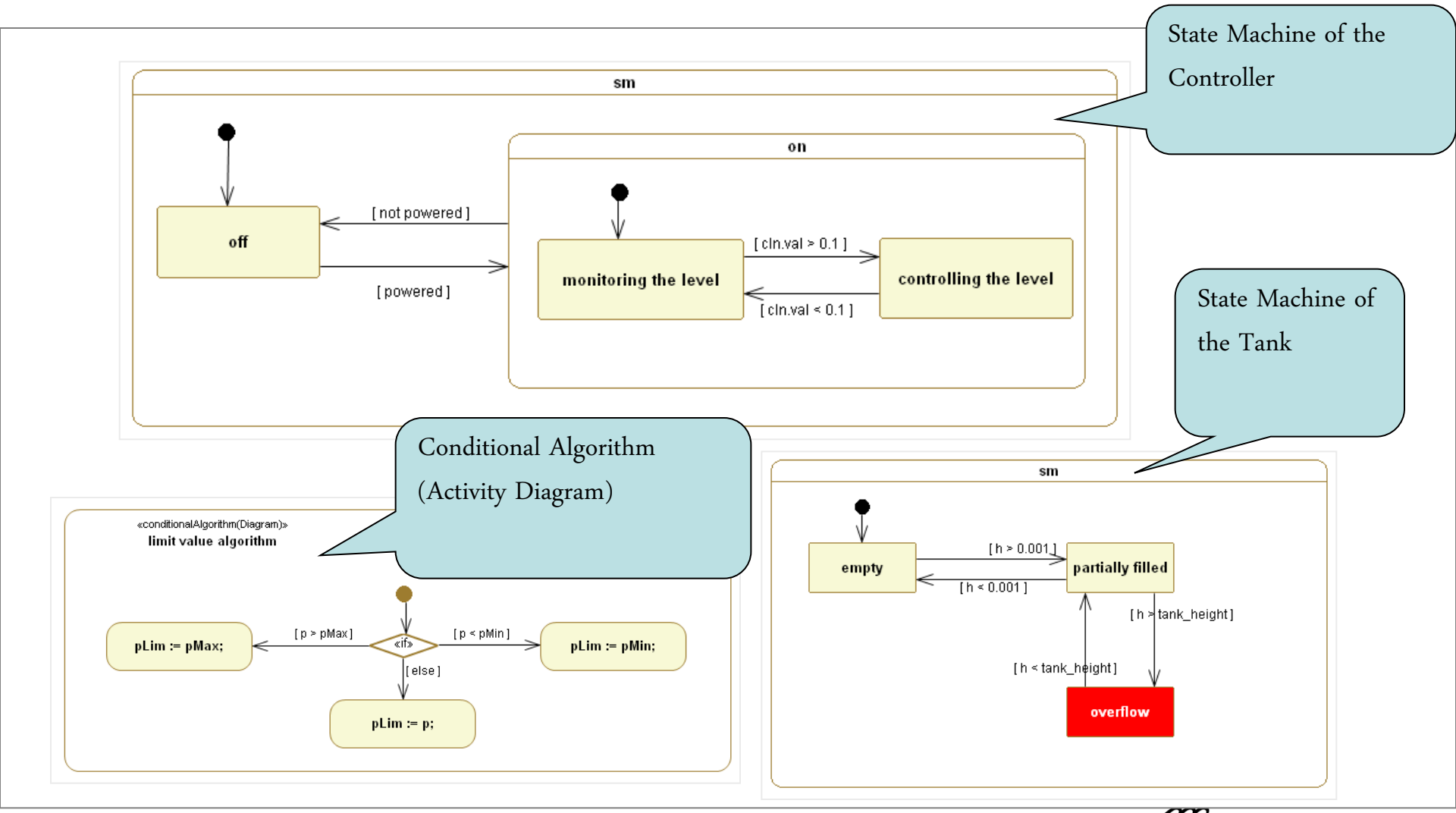


versus

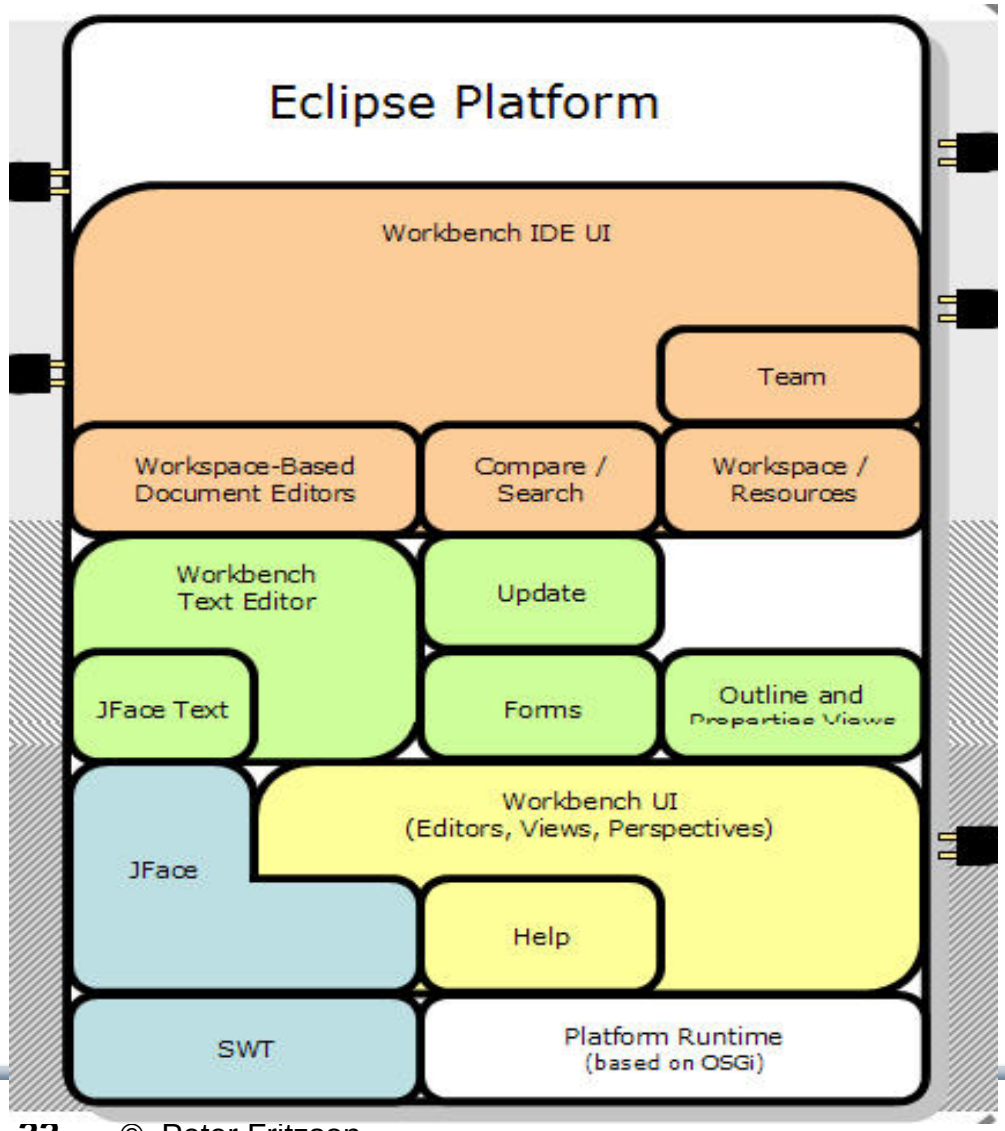


- Modelica Connection diagram
  - Better visual comprehension
  - Predefined connector locations
- Class Internal diagram
  - Nested models
  - Top-model parameters and variables
  - Flow direction
  - Other ModelicaML elements

Example: ModelicaML Representation of System Behavior



The OpenModelica MDT Eclipse Environment for Textual Modeling (I)



Modelica Browser

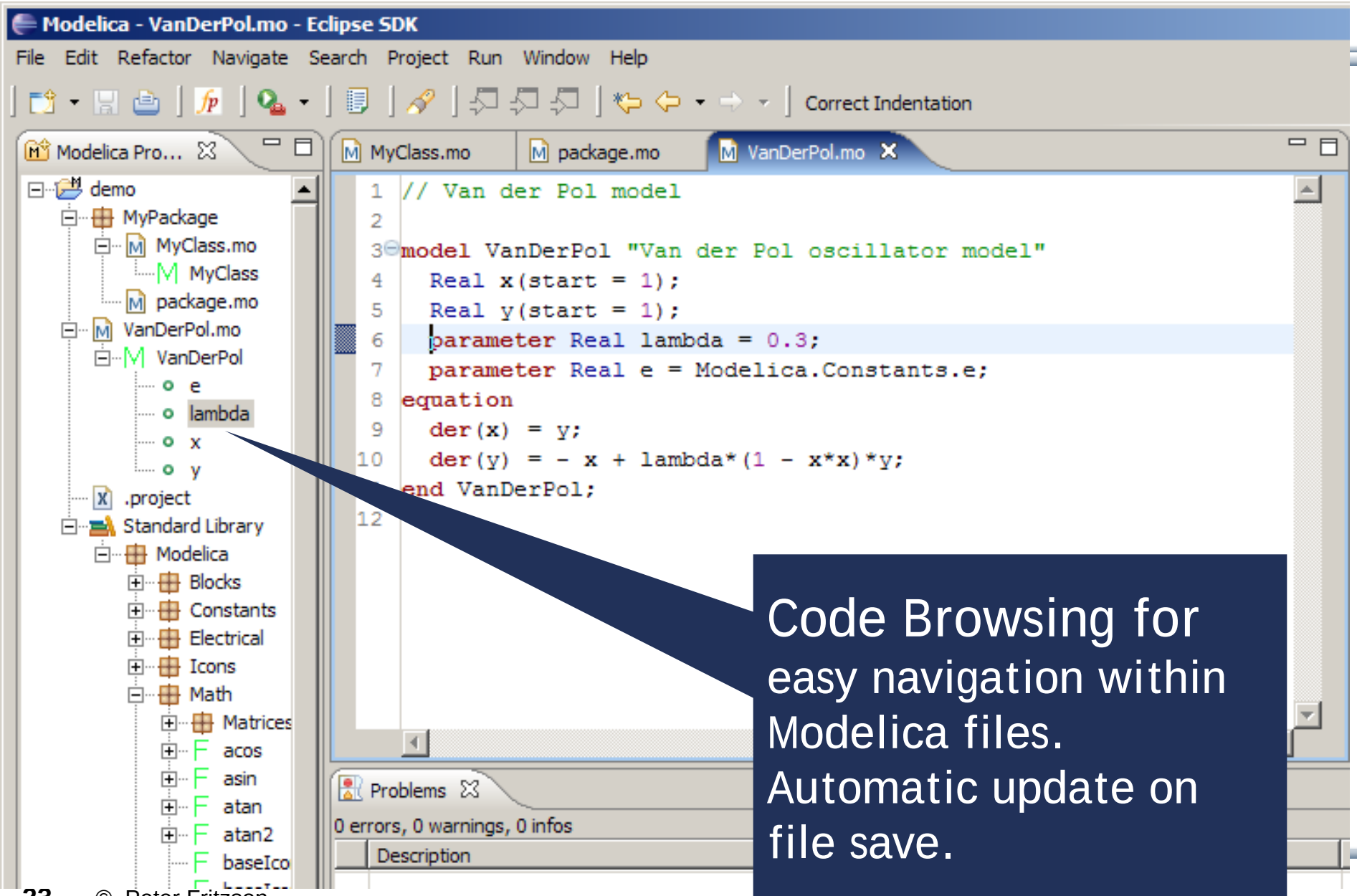
Modelica Editor

Modelica Code Assistant

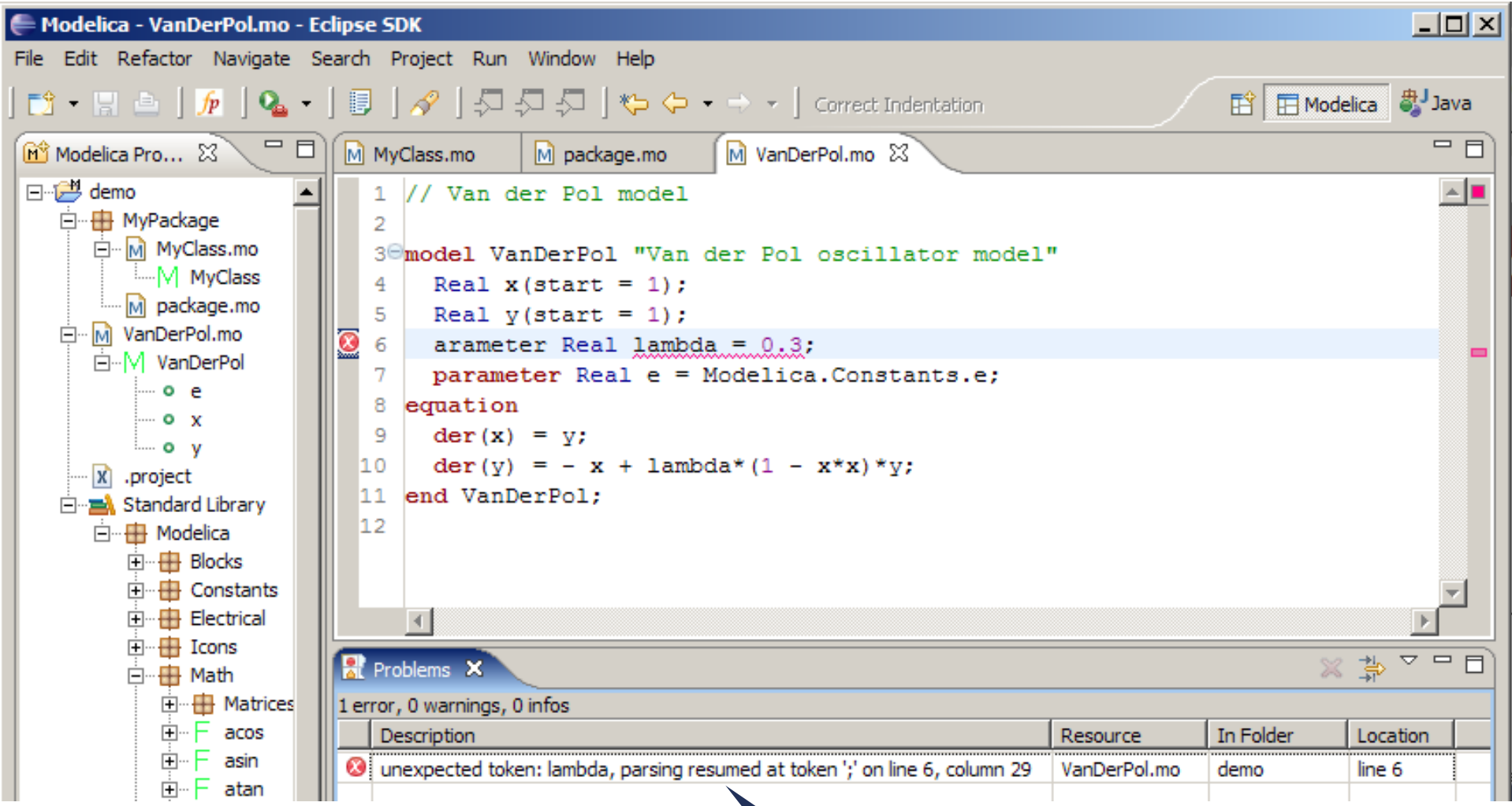
MetaModelica Debugging

Modelica Perspective

MDT: Code Browsing



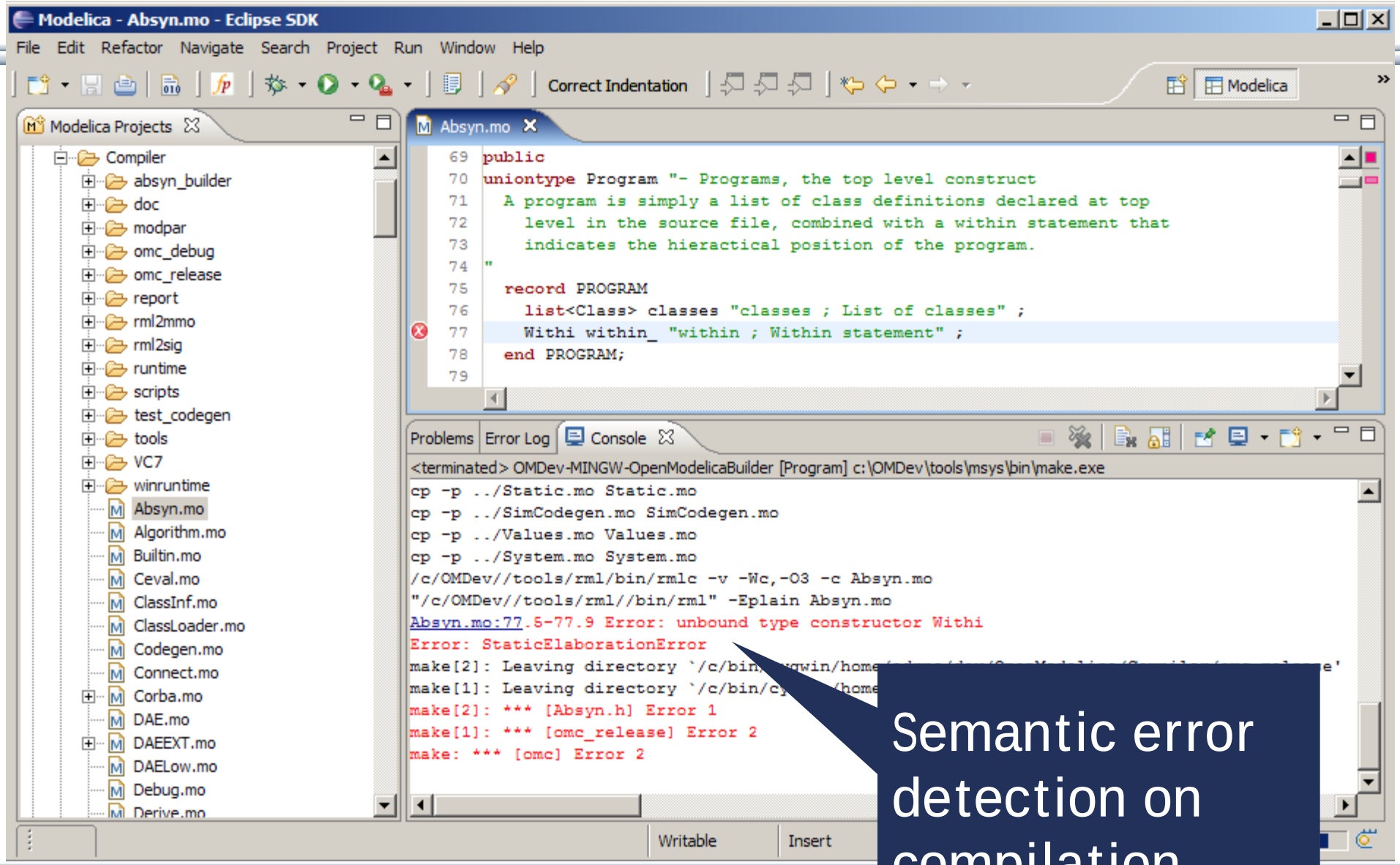
MDT: Parse Error detection



Parse error detection on file save

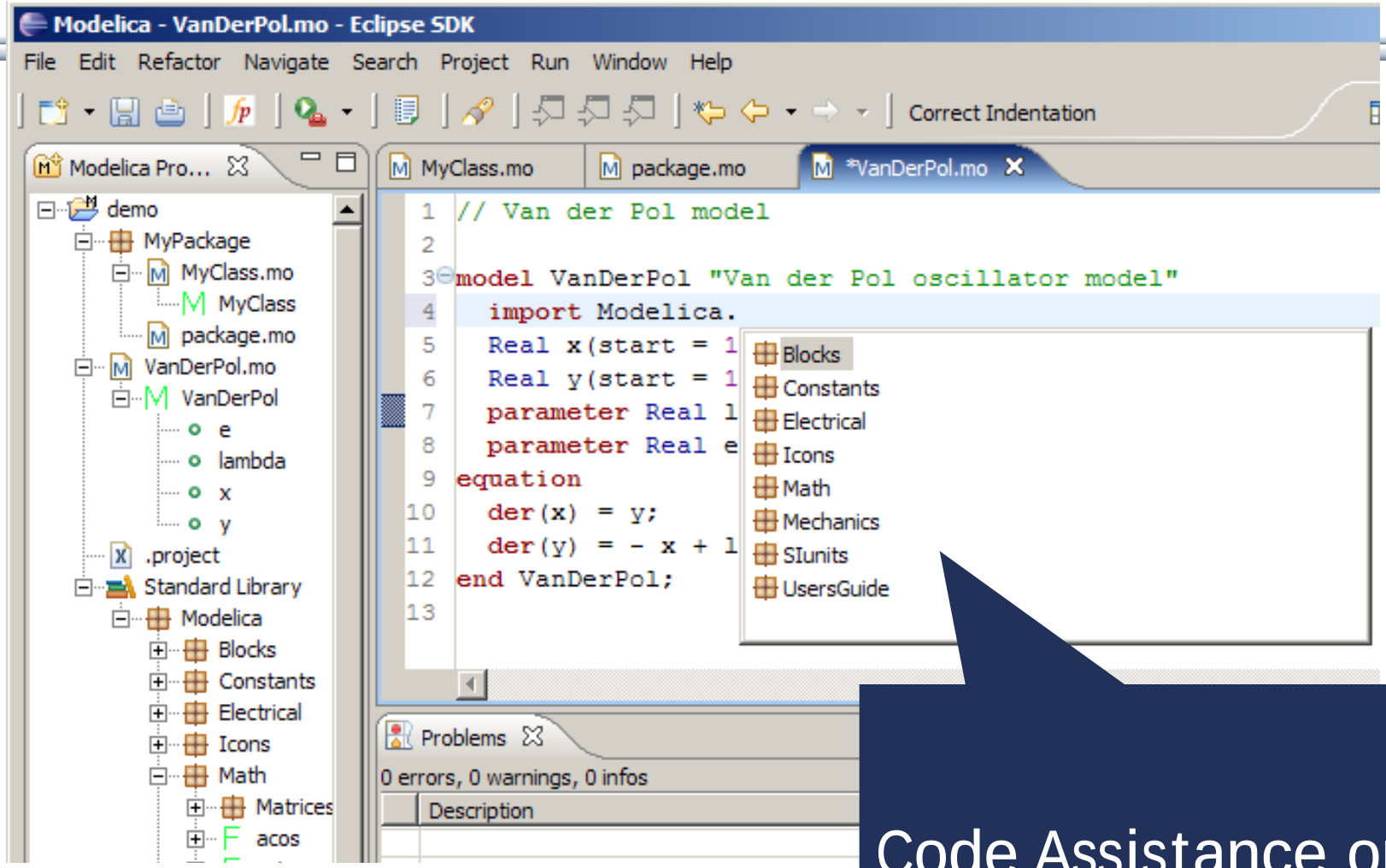
24

MDT: Semantic Error Detection



25

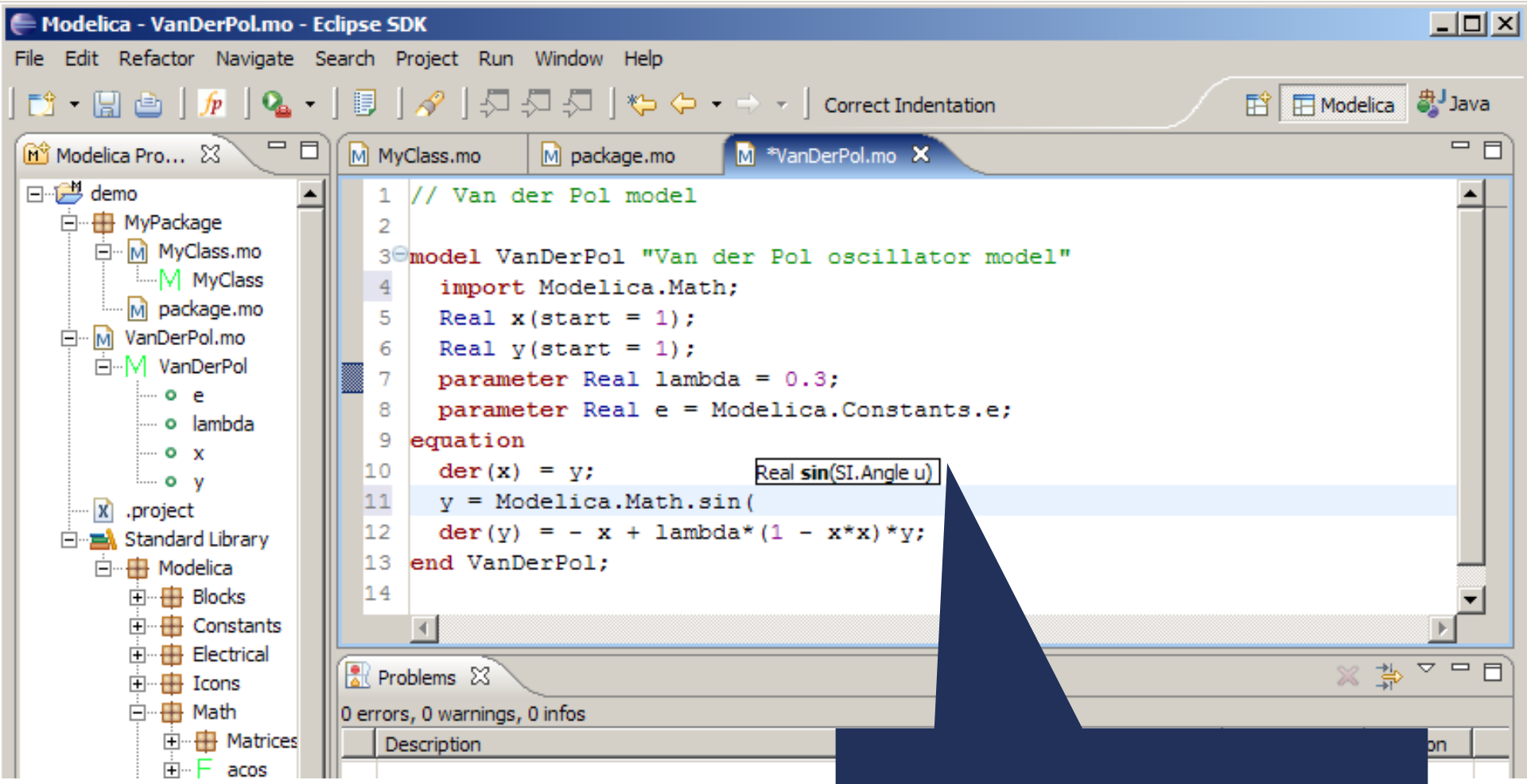
MDT: Code Assistance – on import Statements



Code Assistance on imports

26

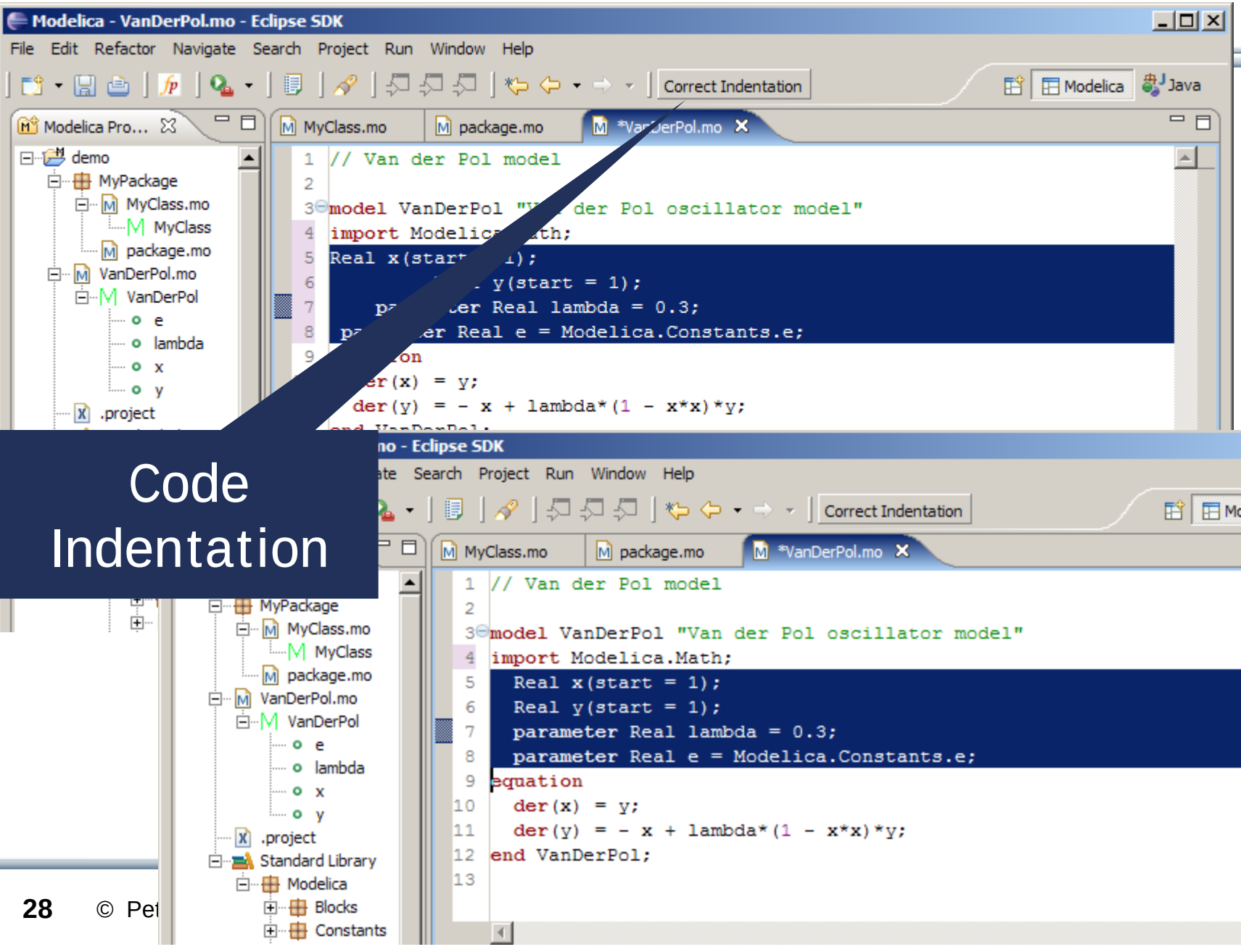
MDT: Code Assistance on Function Calls



Code Assistance on function calls

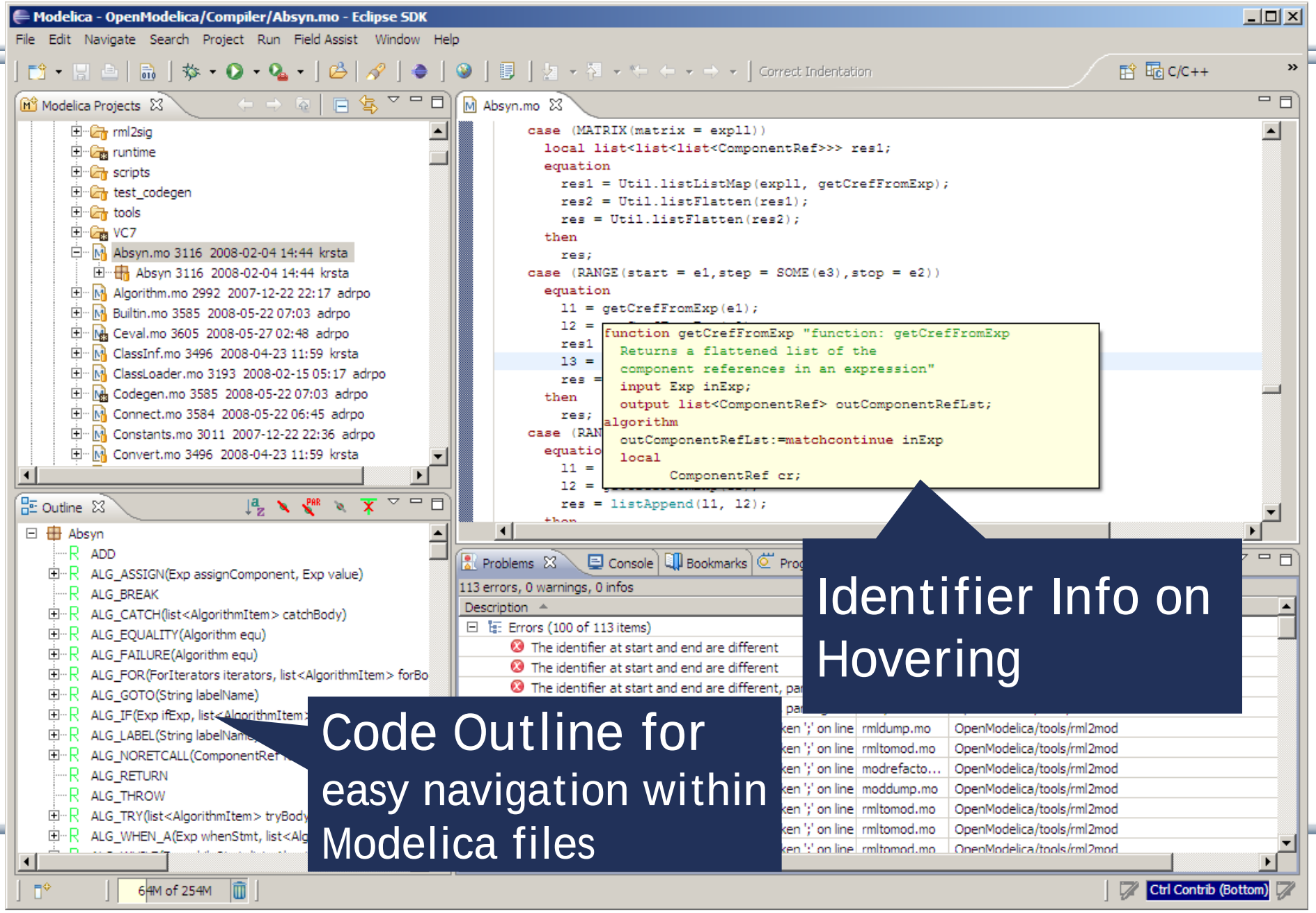
27

MDT: Code Indentation

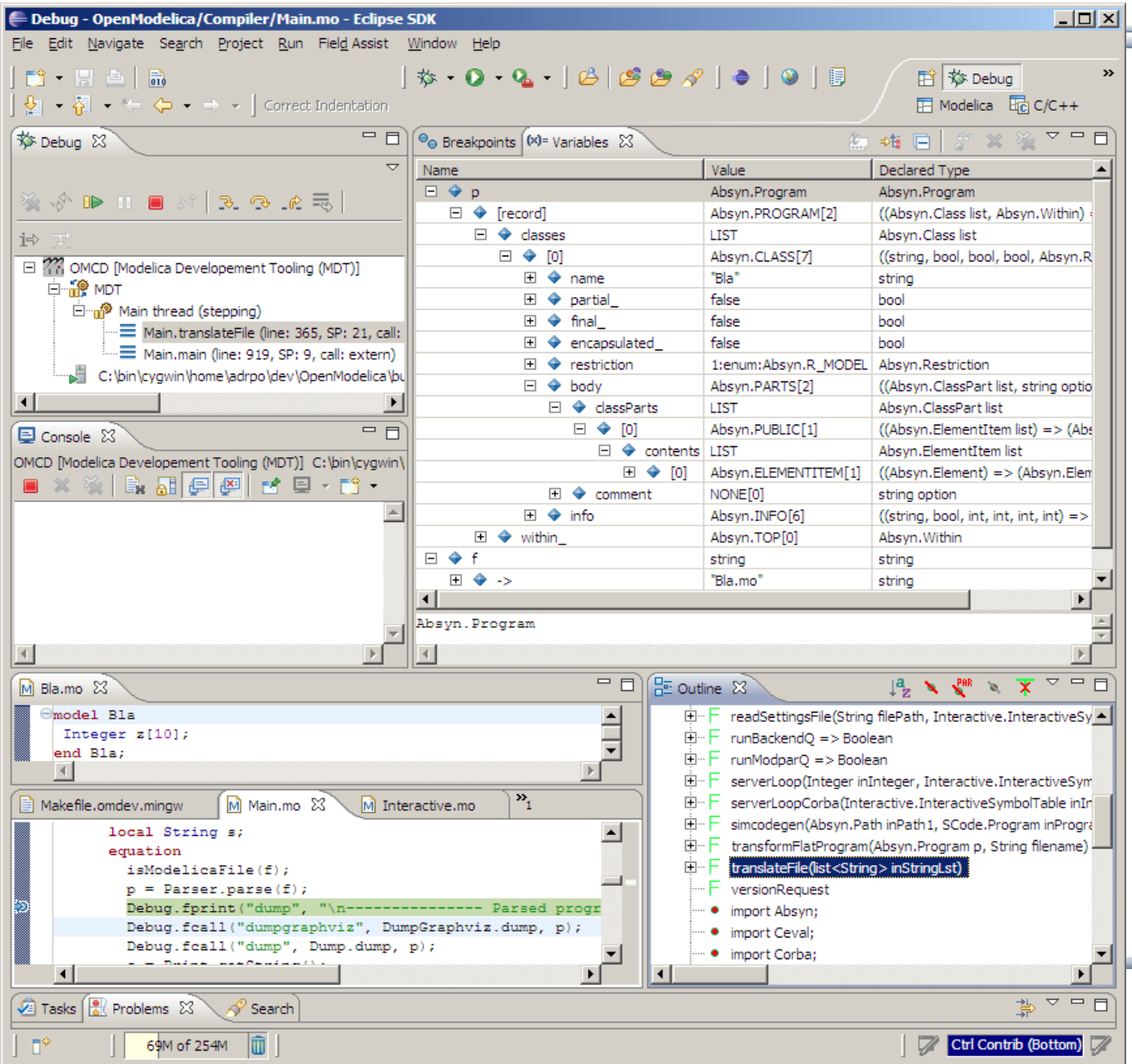


Code Indentation

MDT: Code Outline and Hovering Info

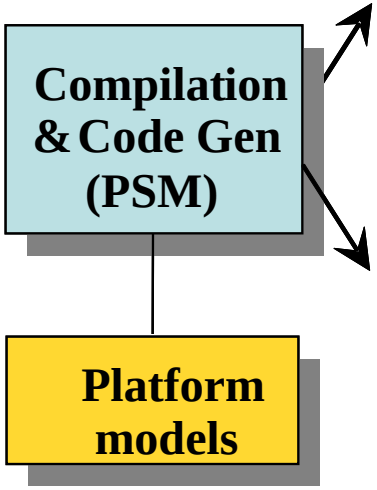


OpenModelica Eclipse MDT Debugging Environment



- Type information for all variables
- Browsing of complex data structures

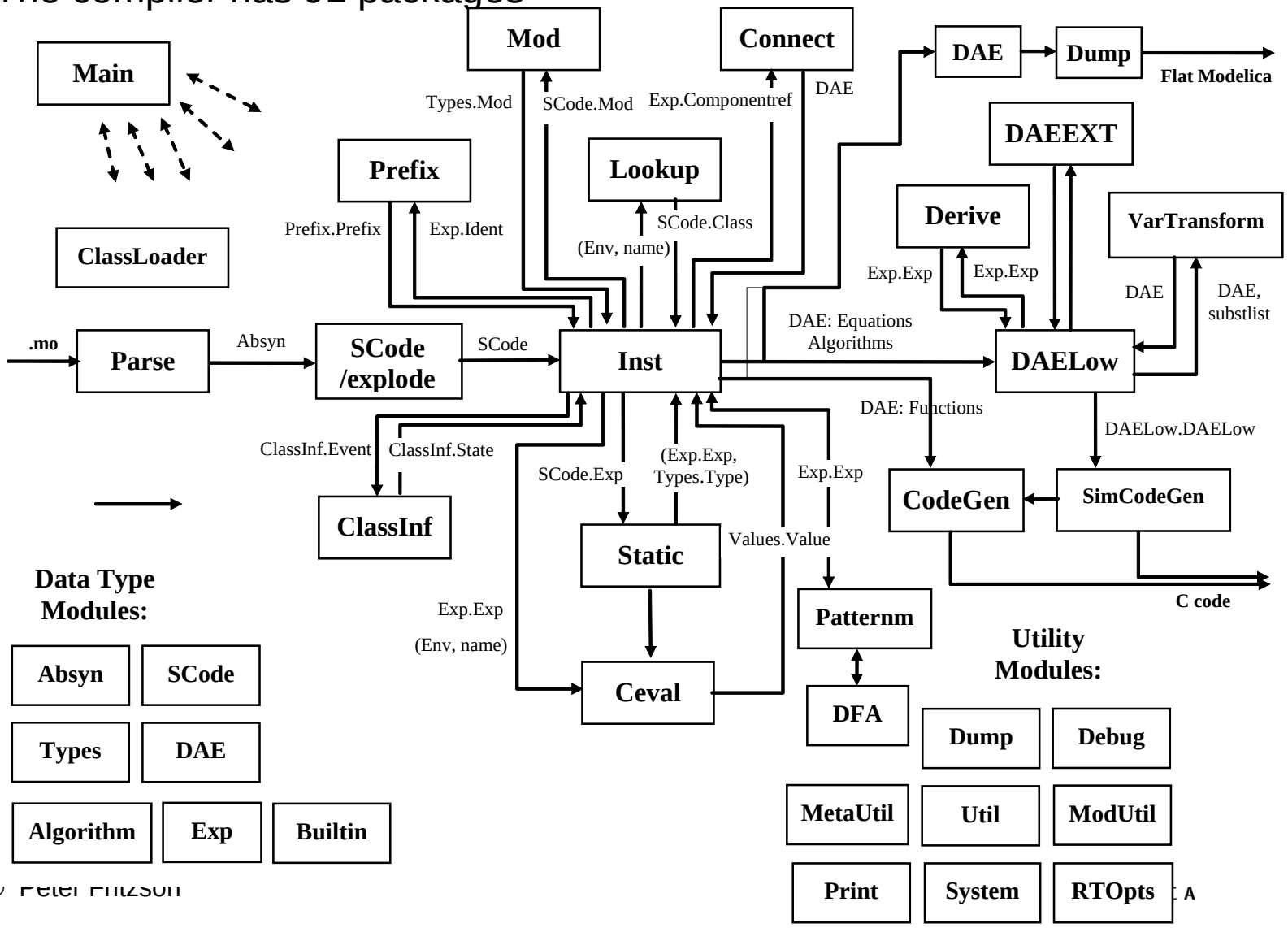
Compilation and Code Generation



- Compilation to C code.
- OpenModelica Text template language for transformation to different platform languages.
- Generation to parallel platforms (Intel multi-core, Nvidia multi-core)

The OpenModelica OMC Compiler – From Modelica to C Code

- Implemented mainly in MetaModelica and C/C++
- The compiler has 91 packages



Template Definition Example
OpenModelica Text Template Language

Used to easily produce differenent code generators, to C, C#, Java, etc.

A text template is a text with holes in it

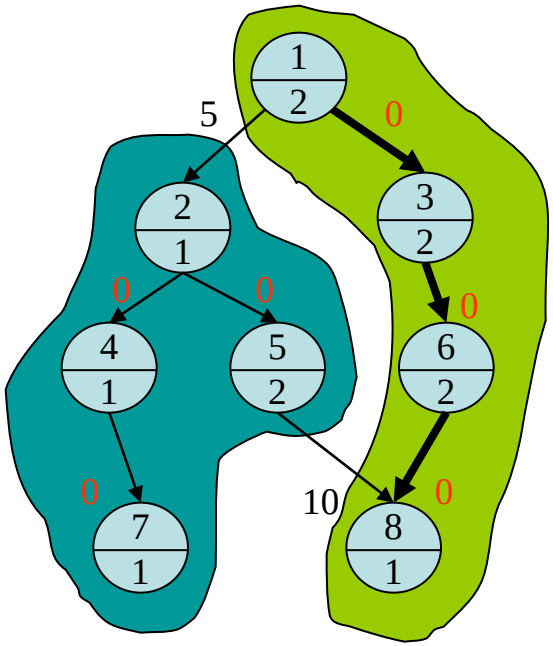
```
hello(String person) ::= <<
Hello <person>!
>>

whileStmt(String cond, list<String> statements) ::=
<<
while(<cond>) {
  <statements \n>
}
>>
```

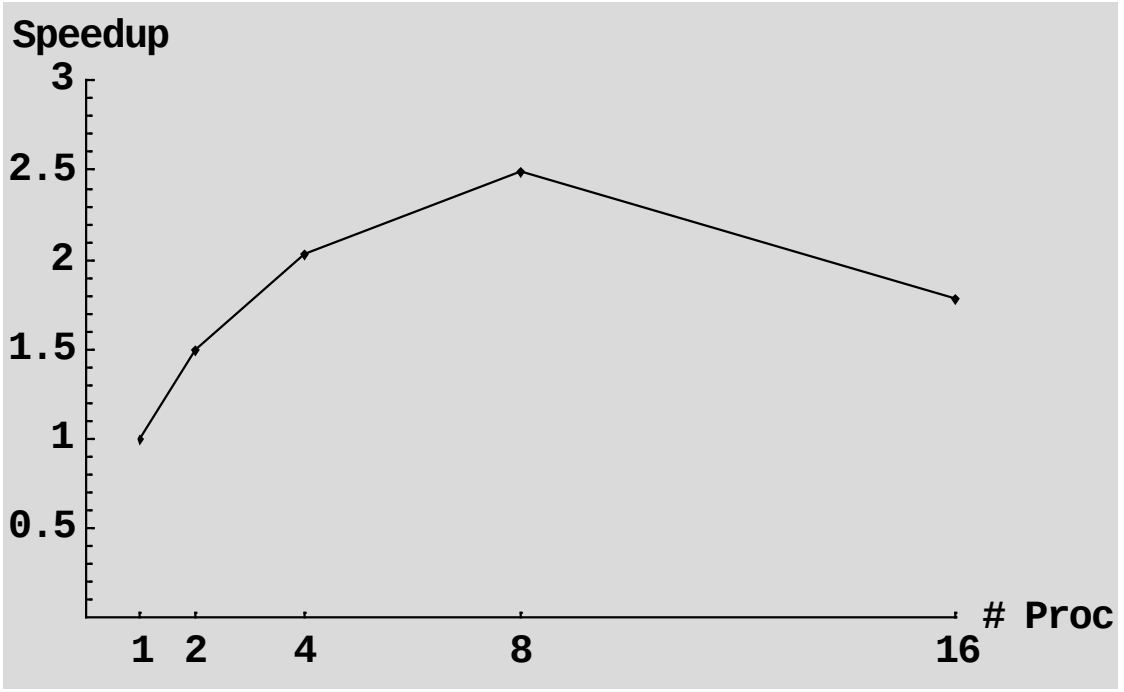
Integrating Parallelism and Mathematical Models Three Approaches

- **Automatic Parallelization of Mathematical Models (ModPar)**
  - Parallelism over the numeric solver method.
  - Parallelism over time.
  - **Parallelism over the model equation system**
    - ... with fine-grained task scheduling
- **Coarse-Grained Explicit Parallelization Using Components**
  - The programmer partitions the application into computational components using strongly-typed communication interfaces.
    - Co-Simulation, Transmission-Line Modeling (TLM)
- **Explicit Parallel Programming**
  - Providing general, easy-to-use explicit parallel programming constructs within the *algorithmic* part of the modeling language.
    - NestStepModelica

Generating Parallel Code from Modelica Task Graphs and Parallelized Application

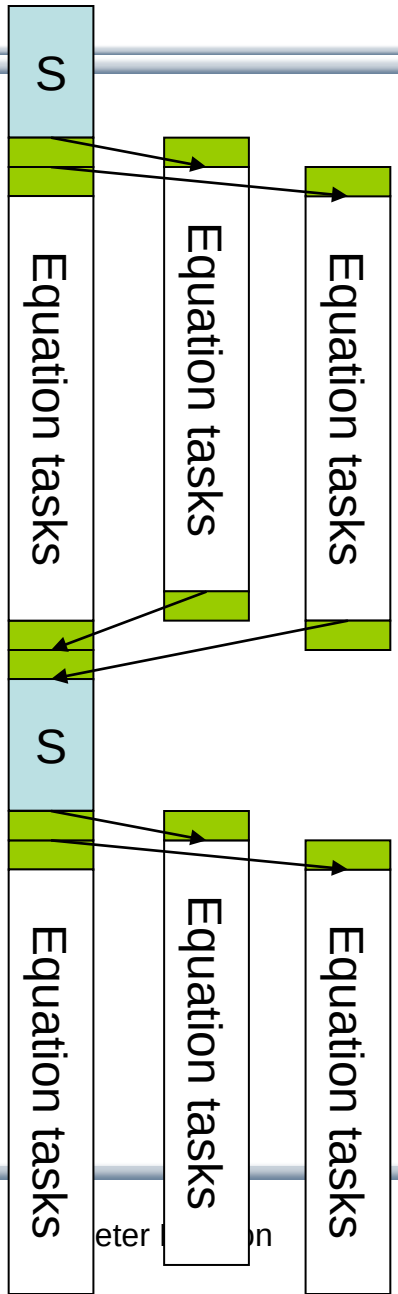


Clustered Task Graph



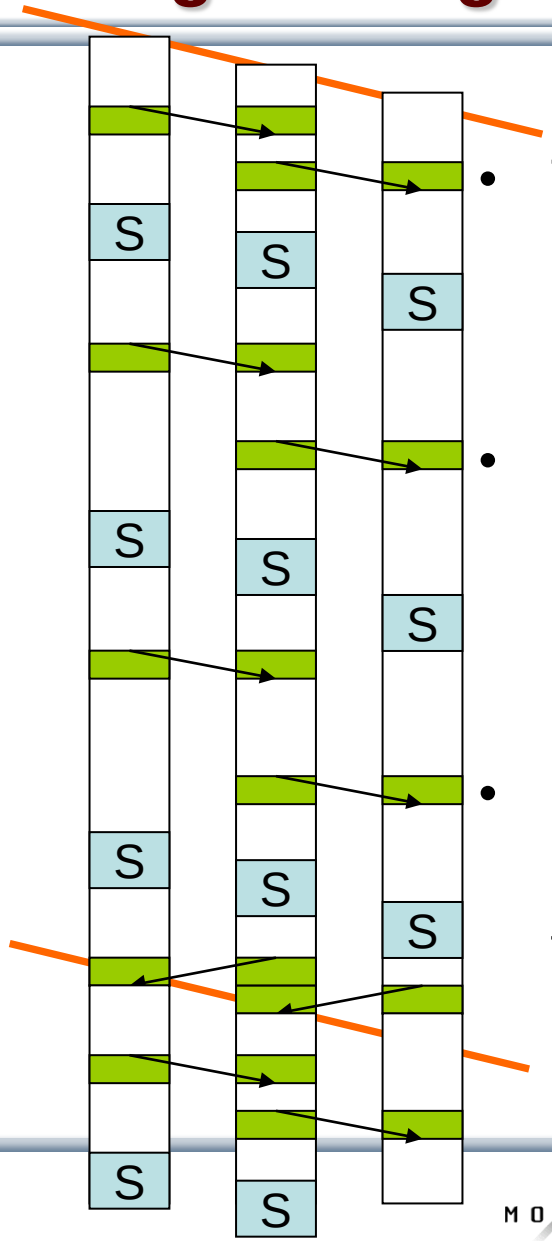
Thermofluid Pipe Application

Task Merging vs New Approach with Pipelining/Inlining



Use a graph rewriting system to merge tasks into larger tasks, based on latency and bandwidth.

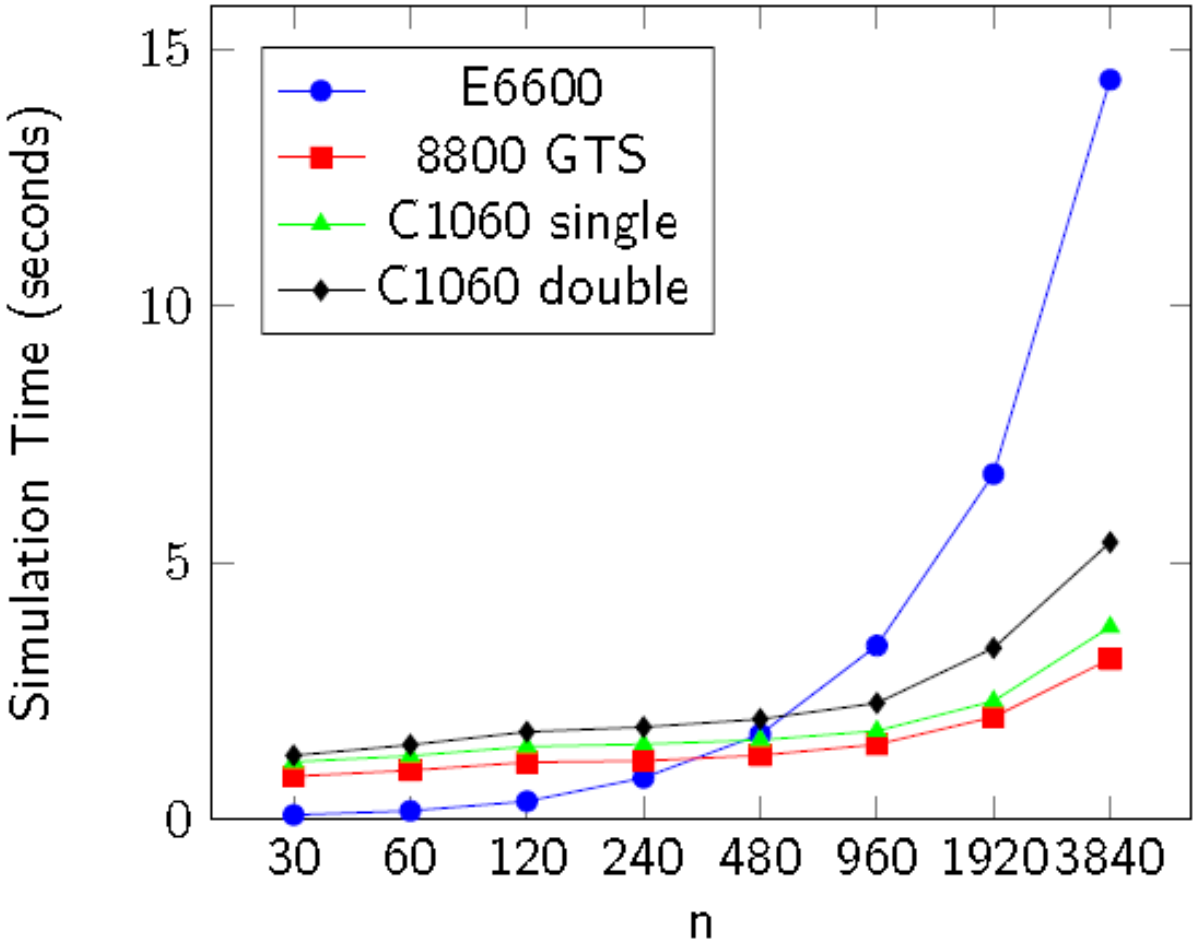
Some tasks are duplicated to avoid communication within a step



- Try to keep communication as close as possible
- Only communicate in one direction inside a time step.
- Solver Inlining – distribute the solver across all the processors

Recent Speedup Measurements on NVIDIA (nov 2009)

Modelica Model, Generated Code, Function of Problem Size



Conclusions

- Businesss process modeling on the way based on System Dynamics
- Several ways of capturing and formalizing requirements in ModelicaML and Behavior Engineering+OpenModelica
- Graphic Modeling (ModelicaML, etc.) and tool support for advanced textual modeling (MDT Eclipse plugin)
- Code generation from OpenModelica, now also using a text template language.
- Parallelization and code generation to parallel platforms