

# The OpenModelica Environment including Static and Dynamic Debugging of Modelica Models and Systems Engineering / Design Verification

LCCC Workshop in Lund

September 19-21, 2012

Peter Fritzson

Professor at Linköping University, Sweden  
 Vice Chairman of Modelica Association  
 Director of Open Source Modelica Consortium  
[peter.fritzson@liu.se](mailto:peter.fritzson@liu.se)

Main Contributors, these topics:

Wladimir Schamai, Martin Sjölund, Adrian Pop, Adeel Asghar  
 & rest of OpenModelica team



A Servo Mechanism Model  
 - a micro example of a full system

1 Introduction

2 DC Motor

3 Simulation Results

4 Conclusions

$$\tau_2 = \frac{1}{k_2} \tau_1$$

$$e = \omega_{ref} - \omega_{out}$$

$$u = K \left( e + \frac{1}{T_I} \int_0^t e \, dt \right)$$

$$v = u \quad u_R = R i \quad u_{emf} = k_1 \omega_{emf}$$

$$v - u_R - u_L - u_{emf} = 0$$

$$u_{emf} = k_1 \omega_{emf} \quad i = \frac{1}{k_1} \tau_{emf} \quad \tau_2 = \frac{1}{k_2} \tau_1$$

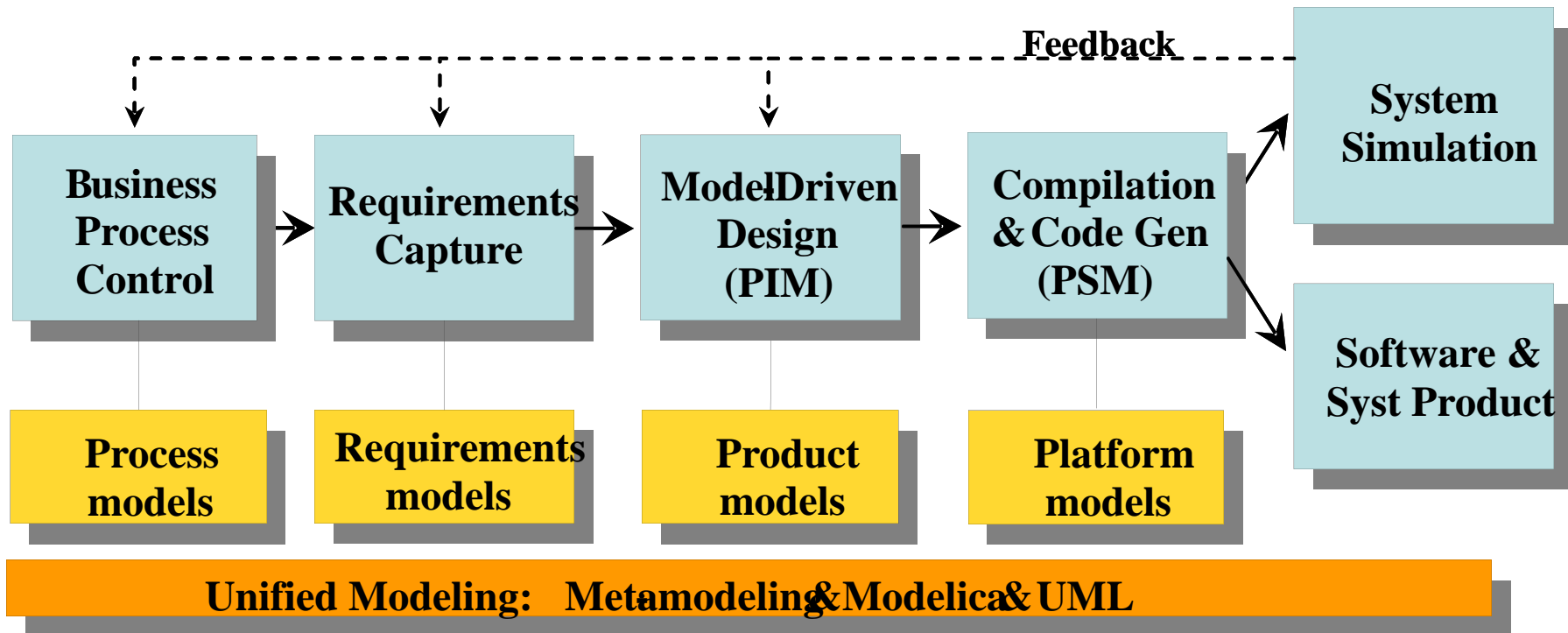
$$\frac{J_1 - J_2 k_2^2}{k_2} \frac{d^2 \theta_2}{dt^2} = \tau_{emf} - k_2 \tau_3$$

# Overview

---

- Background
- Debugging models
- Dynamic verification of requirements

# Vision of Integrated Model-Based Development



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

# Formal Specification of Modelica Static Semantics

---

- First Structured Operational Semantics (SOS)  
Modelica subset formal specification
  - First version 1998, main parts of Modelica static semantics
  - Primarily Big step semantics / Natural Semantics
  - Generating first version of the OpenModelica compiler
- Generating efficient compiler using RML tool
- 2005 converting rule-based syntax into MetaModelica syntax
- 2011 full integration with standard Modelica
  - Bootstrapping of the OpenModelica compiler

# Main Language Extensions

---

- **MetaModelica 2005**
  - Recursive data structures, lists
  - Pattern matching
  - Failure/exception handling, backtracking
- **ParModelica 2011**
  - Dataparallel language constructs, multi-core, e.g. mapping to OpenCL
  - Memory hierarchy for data allocation
- **Optimization extension 2012**
  - Follow same syntax as Optimica in Jmodelica.org
- **ModelicaML extension from 2007**
  - Integrate UML/SysML graphical language and requirement handling
  - Separate tool, not yet integrated in Modelica and the OpenModelica compiler

# OpenModelica – An Open Source Environment

## Open Source Modelica Consortium, 43 org members Aug 2012

Founded Dec 4, 2007

### Open-source community services

- Website and Support Forum
- Version-controlled source base
- Bug database
- Development courses
- [www.openmodelica.org](http://www.openmodelica.org)

### Interactive Modelica compiler (OMC)

- Compiles the Modelica Language
- Modelica and Python scripting

### Environment for creating models

- OMShell – scripting commands
- OMNotebook – interactive notebook
- MDT –Eclipse plug-in
- OMEdit graphic Editor
- OMOptim optimization tool
- ModelicaML UML Profile

The image shows a screenshot of the OpenModelica website and the OMEdit graphic editor. The website, viewed in a browser, has a blue header with the OpenModelica logo and navigation links: HOME, DEVELOPER, FORUM, DOWNLOAD, CONTACT US, WORKSHOP, RESEARCH. It features a 'Top information' section with a 'New OpenModelica website is up.' announcement, a 'Registration' section with a registration form, and an 'Introduction' section. A 'Latest news' sidebar lists recent releases and events. Overlaid on the website is the OMEdit graphic editor, which displays a mechanical model diagram with components like 'world', 'fixed1', 'revolute1', 'body', 'revolute2', 'fixed2', and 'forceAndTorque'. The editor includes a 'Components' panel, a 'Model Browser', and a 'Messages' window.

---

# Debugging Equation-Based Languages and Background

# Problems

---

- Large Gap in Abstraction Level from Equations to Executable Code
- Example error message (hard to understand)

Error solving nonlinear system 132

time = 0.002

residual[0] = 0.288956

x[0] = 1.105149

residual[1] = 17.000400

x[1] = 1.248448

...



# Static vs Dynamic Debugging

---

- **Static Debugging**
  - Analyze the model/program at compile-time
  - Explain inconsistencies and errors, trace error dependencies
  - Example: Underconstrained/overconstrained systems of equations
  - Example: errors in symbolic transformations of models
- **Dynamic Debugging**
  - Find sources of errors at run-time, for a particular execution
  - **Declarative dynamic debugging** – compare the execution with a specification and semi- automatically find the location of the error
  - **Traditional dynamic debugging** – interactively step through the program, set breakpoints, display and modify data structures, trace, stack inspection
- **Goal: Integrated Static and Dynamic Debugging**

# Previous PhD Theses on Dynamic/Static Debugging in Our Group

---

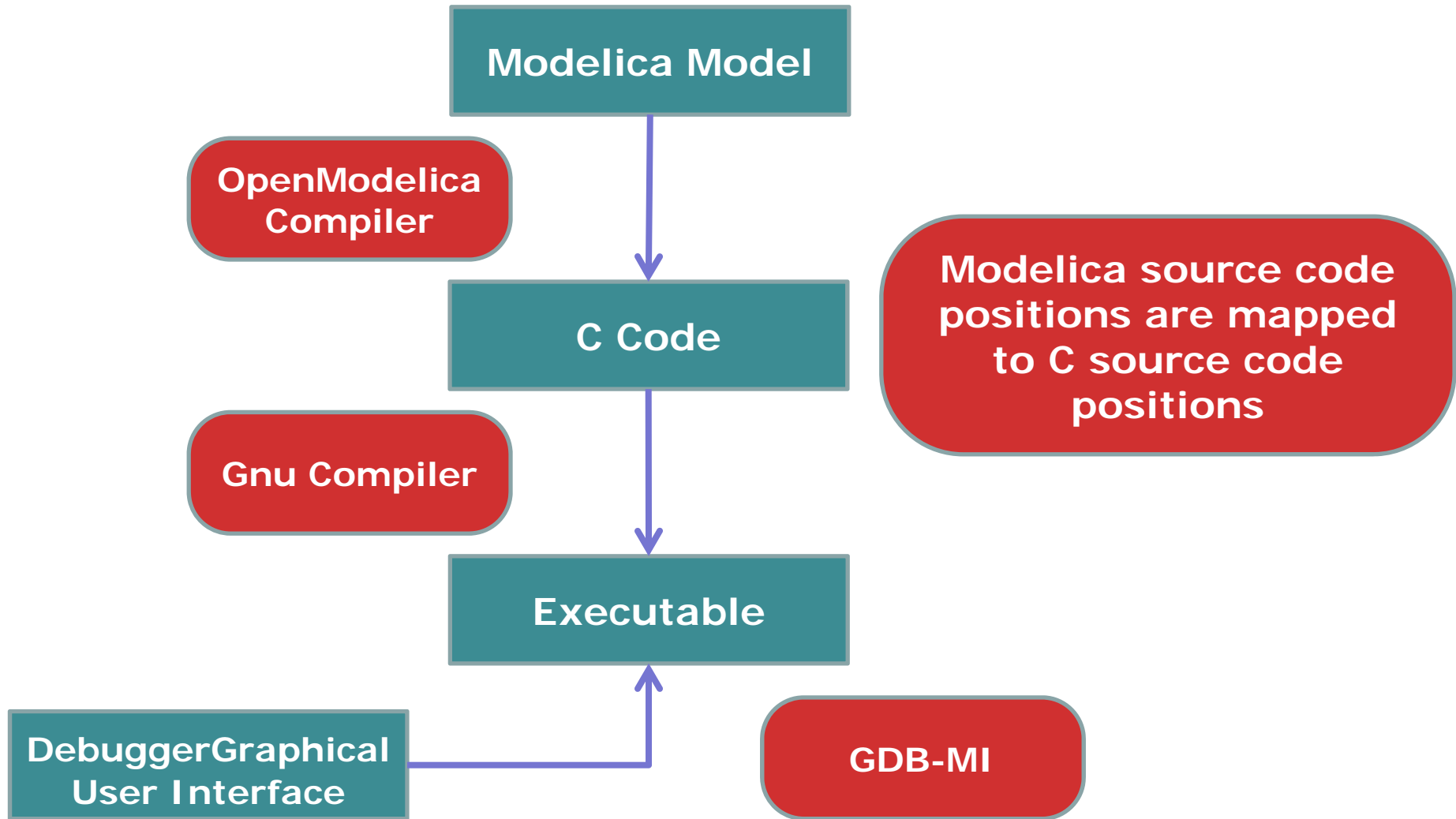
- *Dynamic*. Nahid Shahmeri(1991). Generalized Algorithmic Debugging
- *Dynamic*. Mariam Kamkar(1993). Interprocedural Dynamic Slicing with Applications to Debugging and Testing
- *Dynamic*. Henrik Nilsson(1998). Declarative Debugging for Lazy Functional Languages
- *Static/Dynamic*. Peter Bonus (June 2004). Debugging Techniques for Equation-Based Languages.
- *Dynamic*. Adrian Pop (June 5, 2008). Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages

---

# Dynamic Debugging

## Large Modelica Algorithmic Code Models

# Tool Architecture and Communication



# Example Mapping Modelica Postions to C Code

Convert Modelica code to C source code by adding Modelica line number references.



```

M HelloWorld.mo X
1 function HelloWorld
2   input Real x;
3   output Real y;
4   algorithm
5     y := sin(x);
6   end HelloWorld;

```

```

.c HelloWorld.conv.c X
57 #line 29 "HelloWorld.c"
58 /* functionBodyRegularFunction: var inits */
59 #line 30 "HelloWorld.c"
60 /* functionBodyRegularFunction: body */
61 #line 5 "/c/workspace/HelloWorld/HelloWorld.mo"
62   tmp2 = sin(_x);
63 #line 5 "/c/workspace/HelloWorld/HelloWorld.mo"
64   _y = tmp2;
65 #line 35 "HelloWorld.c"

```

# Debugger Integrated in Eclipse OpenModelica MDT Environment

- Eclipse plugin **MDT (Modelica Development Tooling)** is the integrated development environment
- Debugger is a debug plug-in within MDT

The screenshot displays the Eclipse IDE with the MDT GDB debugger. The main window is titled "Debug - trunk/Compiler/Script/Interactive.mo - Eclipse SDK". The interface is divided into several panes:

- Stack Frame List:** Located on the left, it shows the call stack. The current frame is "Main Thread (stepping)" at line 2034 of "Interactive.mo". Other frames include "evaluateGraphicalApi at Interactive.mo:2034", "evaluate2 at Interactive.mo:485", "evaluateToStdOut at Interactive.mo:328", "translateFile at Main.mo:619", "bcall at Debug.mo:460", and "main at Main.mo:1183".
- Variables View:** Located in the top right, it shows a table of variables. The table has columns for Name, Declared Type, Value, and Actual Type. The variable "loadModel" is highlighted, showing its value as "loadModel" and its type as "void \*".
- Code Editor:** The bottom left pane shows the source code of "Interactive.mo". The current line is highlighted, showing a call to "matchApiFunction".
- Output View:** Located at the bottom right, it shows the output of the debugger. The output text includes "MDT GDB [Modelica Development Tooling (MDT) GDB] C:\OpenModelica\trunk\testsuite\bootstrapping\main.exe".

Red boxes highlight the Stack Frame List, the Variables View, and the Output View. Red text labels "List of Stack Frames", "Variables View", and "Output View" are placed below their respective panes.

---

# Static Debugging

## Transformational Debugging of Equation-Based Models

# Debugging Equation Systems

---

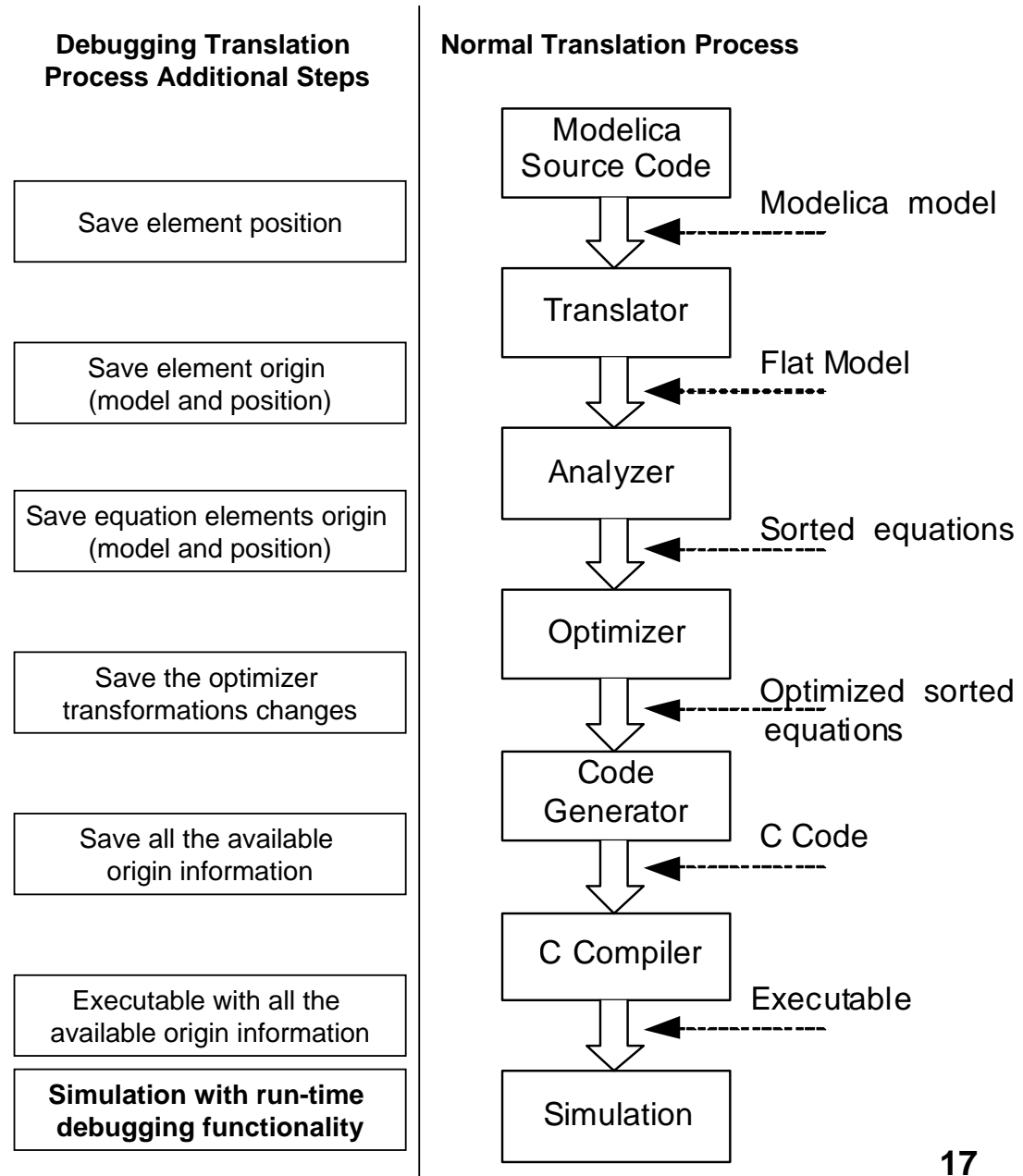
## Modelica Compiler Backend

- Complex mathematical transformations
- Hidden to users
- Users want to access this information
- Not intuitive, because
  - No explicit control flow
  - Numerical solvers
  - Linear/Non-linear blocks
  - Optimization
  - Events



# Translation Phases with Model Debugging

- **Include debugging support within the translation process**



# Input to Debugger: Modelica Model

---

```
class RC // 24 equations and variables
...
equation
...
ground1.p.v = 0.0;
0.0 = resistor1.p.i + resistor1.n.i;
resistor1.i = resistor1.p.i;
resistor1.T_heatPort = resistor1.T;
capacitor1.i = capacitor1.C * der(capacitor1.v);
capacitor1.v = capacitor1.p.v - capacitor1.n.v;
0.0 = capacitor1.p.i + capacitor1.n.i;
capacitor1.i = capacitor1.p.i;
...
end RC;
```

# Output from Compiler Frontend: Sorted ODE or DAE (Differential Algebraic Equations)

```
class RC // 24 equations and variables
```

```
...
```

```
equation
```

```
...
```

```
ground1.p.v = 0.0;  
0.0 = resistor1.p.i + resistor1.n.i;  
resistor1.i = resistor1.p.i;  
resistor1.T_heatPort = resistor1.T;  
capacitor1.i = capacitor1.C *  
  der(capacitor1.v);  
capacitor1.v = capacitor1.p.v -  
  capacitor1.n.v;  
0.0 = capacitor1.p.i + capacitor1.n.i;  
capacitor1.i = capacitor1.p.i;
```

```
...
```

```
end RC;
```

```
class RC // 5 equations and variables
```

```
...
```

```
// 14 alias variables 5 constants
```

```
equation
```

```
  sinevoltage1.signalSource.y =  
sinevoltage1.signalSource.offset + (if time <  
sinevoltage1.signalSource.startTime then 0.0  
else sinevoltage1.signalSource.amplitude *  
sin(6.28318530717959 *  
(sinevoltage1.signalSource.freqHz * (time -  
sinevoltage1.signalSource.startTime)) +  
sinevoltage1.signalSource.phase));  
  resistor1.v = capacitor1.v -  
sinevoltage1.signalSource.y;  
  capacitor1.i = -resistor1.v / resistor1.R_actual;  
  resistor1.LossPower = -resistor1.v *  
capacitor1.i;  
  der(capacitor1.v) = capacitor1.i / capacitor1.C;  
end RC;
```

# Symbolic Transformations

---

- From source code to flat equations
  - Most of the structure remains
  - Few symbolic manipulations (mostly simplification/evaluation)
- Equation System Optimization
  - Changes structure
  - Strong connected components
  - Variable replacements
  - ... and more

# Tracing Symbolic Transformations

---

- Simple Idea
  - Store transformations as equation metadata
- Works best for operations on single equations
  - Alias Elimination ( $a = b$ )
  - Equation solving ( $f_1(a,b) = f_2(a,b)$ , solve for  $a$ )
- Multiple equations require special handling
  - Gaussian Elimination (linear systems, several equations)
  - ...

# Tracing Overhead?

---

- OpenModelica compiler implementation is so fast that tracing is enabled by default
  - 1 extra comparison and/or cons operation per optimization
  - Not noticeable during normal compilation
  - Less than 1% time overhead for tracing
- No real overhead unless you output the trace

# Substitution Example, Storing the Trace

---

$$a = b$$

$$c = a + b$$

$$d = a - b$$

$$c = a + b \text{ (subst } a=b) \Rightarrow$$

$$c = b + b \text{ (simplify)} \Rightarrow$$

$$c = 2 * b$$

$$d = a - b \text{ (subst } a=b) \Rightarrow$$

$$d = b - b \text{ (simplify)} \Rightarrow$$

$$d = 0.0$$

- The alias relation  $a=b$  stored in variable  $a$
- The equations are e.g. stored as  $(lhs,rhs,list<ops>)$

# Debugging Using the Transformation Trace

---

- Text output
  - Initial implementation
  - Verify performance and correctness of the trace
- Structured output based on database storage
  - Graphical debugging
  - Cross-referencing equations (dependents/parents)
  - Ability to see why a variable is solved in a particular way
  - Requires a schema
  - Future work/work in progress



# Trace Example (1)

---

$$0 = y + \text{der}(x * \text{time} * z); \quad z = 1.0;$$

**(1) substitution:**

$$y + \text{der}(x * (\text{time} * z))$$

$\Rightarrow$

$$y + \text{der}(x * (\text{time} * 1.0))$$

**(2) simplify:**

$$y + \text{der}(x * (\text{time} * 1.0))$$

$\Rightarrow$

$$y + \text{der}(x * \text{time})$$

**(3) expand derivative  
(symbolic diff):**

$$y + \text{der}(x * \text{time})$$

$\Rightarrow$

$$y + (x + \text{der}(x) * \text{time})$$

**(4) solve:**

$$0.0 = y + (x + \text{der}(x) * \text{time})$$

$\Rightarrow$

$$\text{der}(x) = ((-y) - x) / \text{time}$$

## Trace Example (2)

---

**differentiation:**

$$d/dtime L ^ 2.0$$

=>

$$0.0$$

**differentiation:**

$$d/dtime x ^ 2.0 + y ^ 2.0$$

=>

$$2.0 * (der(x) * x + der(y) * y)$$

**Substitution:**

$$2.0 * (der(x) * x + der(y) * y)$$

=>

$$2.0 * (\$DER.x * x + \$DER.y * y)$$

=>

$$2.0 * (u * x + \$DER.y * y)$$

=>

$$2.0 * (u * x + v * y)$$

=>

$$2.0 * (u * xloc[1] + v * xloc[0])$$

# Readability of Transformation Trace

- Most equations have very **few** transformations on them
- Most of the interesting equations have a few
  - Still rather readable
- Some extra care to handle Modelica variable aliasing

## MSL 3.1 MultiBody DoublePendulum

# Ops	Frequency	Comment
0	457	Parameters
1	89	Dummy eq & know var
2	720	Alias vars
3	479	Alias vars
4	124	Alias after simplify
5	25	Alias after simplify
6	99	Alias after simplify
7	55	Scalar eq
8	37	...
9	110	...
10	72	...
11	12	...
12	25	...
13	35	...
14	3	Known constant after many replacements
21	27	World object (3x3 matrix with many occurrences of aliased vars)

# Future Work on Transformational Debugging

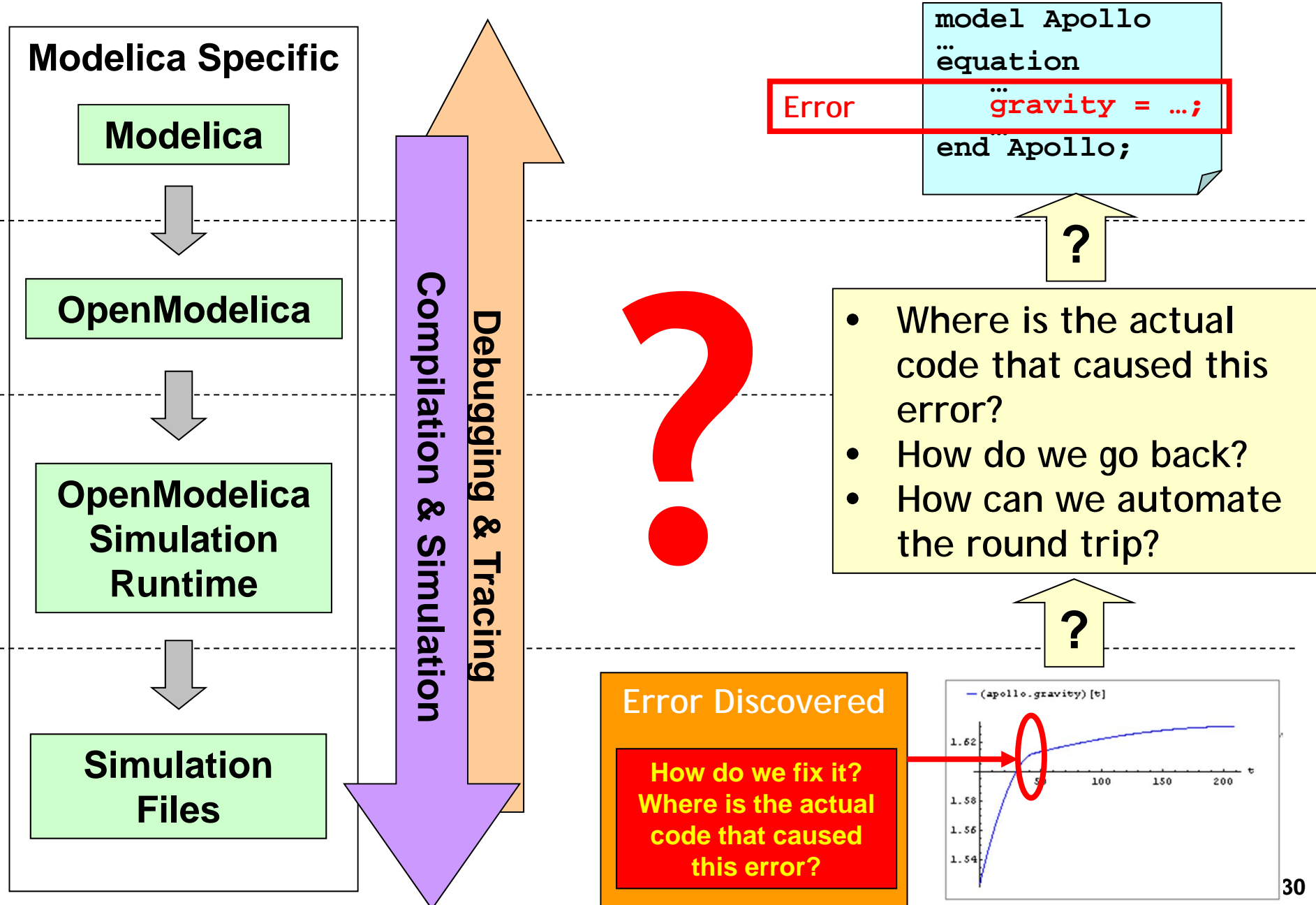
---

- Structural debug information queries based on a database
- Graphical debugger
- Simulation runtime uses database
- More operations recorded
  - Dead code elimination
  - Control flow and events
  - Forgotten optimization modules

---

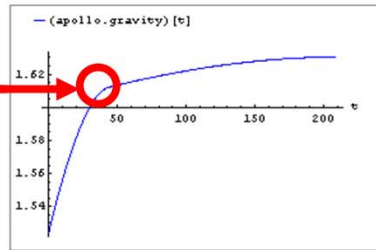
# Integrated Debugging

# Need to Combine Approaches to Help the User



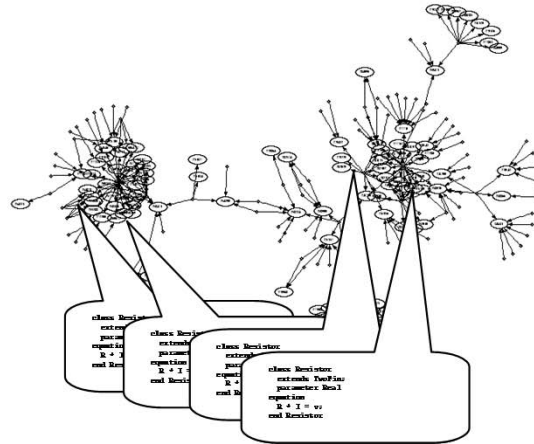
# Integrated Debugging Approach

**Error Discovered**  
**What now?**  
*Where is the equation or code that generated this error?*



**Build graph**

**Interactive Dependency Graph**  
*These equations contributed to the result*



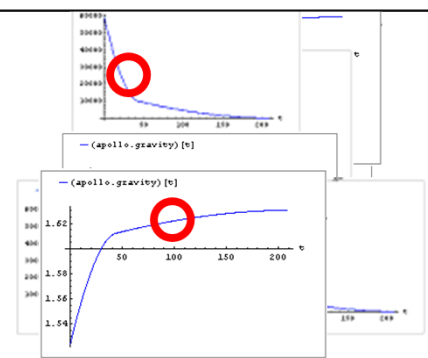
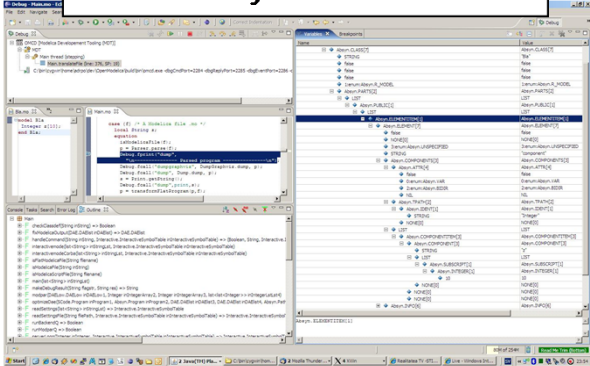
**Code viewer**  
*Show which model or function the equation node belongs to*

**Follow if error is in a function**

**Follow if error is in an equation**

**Algorithmic Code Debugging**  
*Normal execution point debugging of functions*

**Simulation Results**  
*These are the intermediate simulation results that contributed to the result*



- Mark the error
- Build an interactive graph from the transformation trace
- Walk the graph interactively to find the error

# Debugging Based on User Interaction

---

- The interactive dependency graph contains two types of edges:
  - *Calculation dependency edges*
  - *Origin edges from traced symbolic transformations*
- The user interacts with the dependency graph in several ways:
  - *Displaying simulation results* through selection of the variables
  - *Classifying a variable* as having wrong values
  - *Classifying an equation* as correct
  - *Building a new dependency graph* based on the new set of variables with wrong values (classified variables) or by modifying the equations or parameter values nodes.
  - *Displaying model code* by following origin edges
  - *Invoking the algorithmic code debugging subsystem*



# Debugging Summary

---

- Debugging **equation-based** models present new **challenges**
- **Equation** systems are **transformed** symbolically to a form hard for the user to recognize
- Static **transformational** debugging **explains** the transformations and maintains a mapping between the low level and the high level model
- **Dynamic debugging** helps to **walk** through a model/program and **inspect** data for an execution
- **Goal: integrated static/dynamic debugging approach**

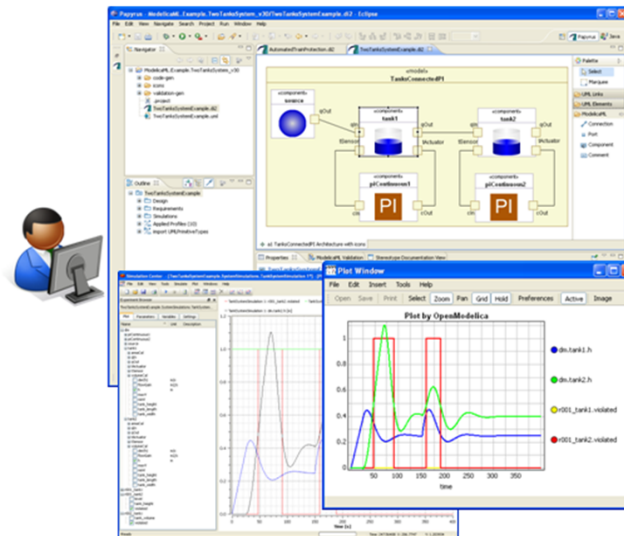
---

# Requirements traceability and dynamic model verification

# Introduction: ModelicaML Background

- **ModelicaML Eclipse plug-in Modelica/UML profile** integrates a subset of the UML and the Modelica language in order to leverage standardized graphical notations of UML for system modeling and the simulation power of Modelica
- ModelicaML enables engineers to describe
  - System **requirements**
  - System **design** (structure and behavior)
  - Usage-, test **scenarios**
- **vVDR (Virtual Verification of Designs against Requirements)** is a method that enables a model-based design verification against requirements
- **vVDR** is supported in ModelicaML

① System Modeling with ModelicaML



② Modelica Code Generation

```
within TestTask;
function TestTask
input Real pIn;
output Real pOut;
output Real p;
input Real pi;
end TestTask;

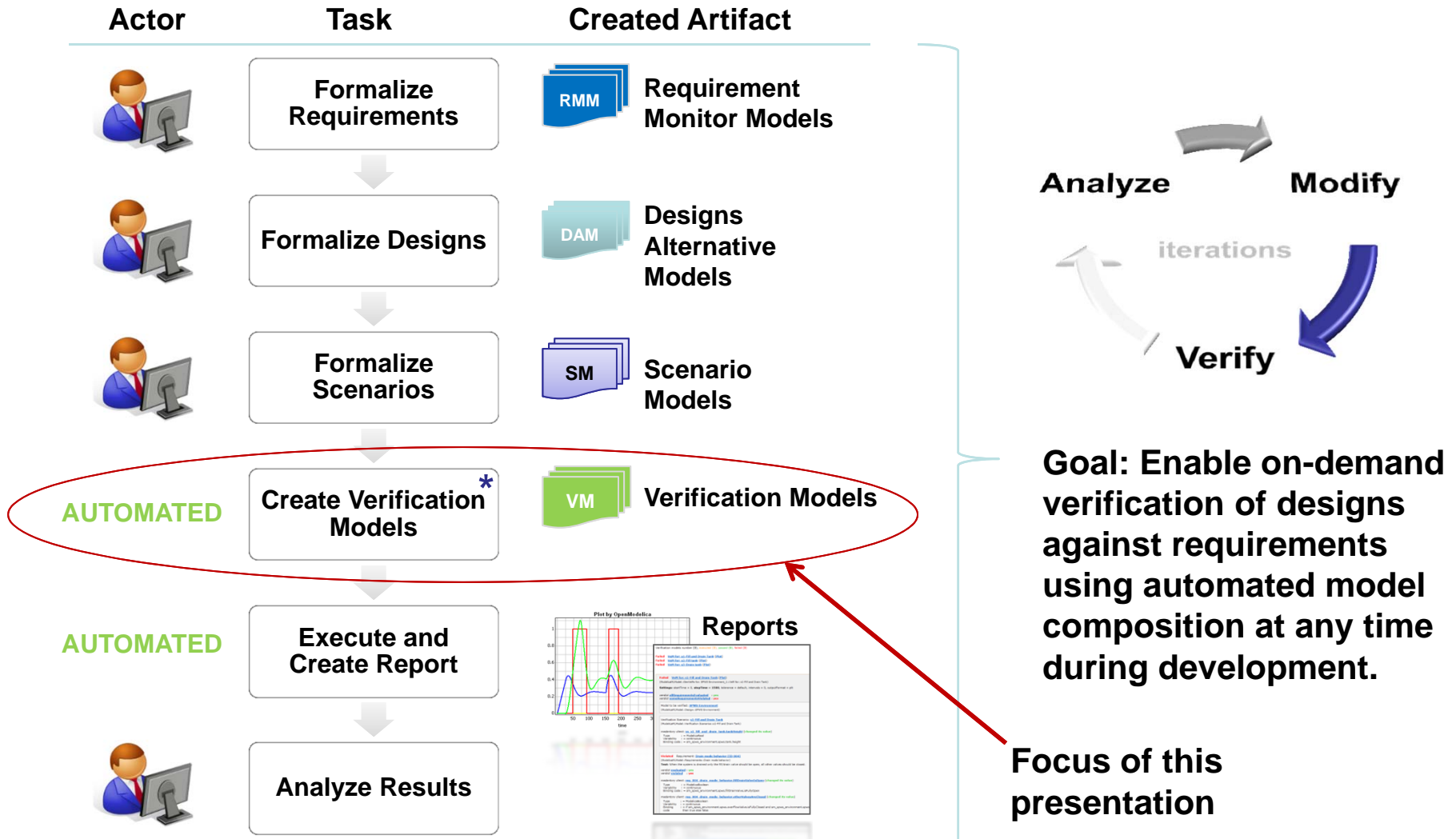
algorithm
// code generated from the activity "TestTask Diagram"
if pi > 0 then
  p := pi;
else
  p := 0;
end if;
end TestTask;

within TestTaskControl;
model TestTaskControl
parameter Real gain = 1; // Gain for control loop
parameter Real length = 10; // Length of control loop
parameter Real max = 1; // Max value for control loop
parameter Real min = 0; // Min value for control loop
parameter Real tau = 0.1; // Time constant for control loop
parameter Real tauInt = 0.1; // Integration time constant
parameter Real tauDiff = 0.1; // Differentiation time constant
parameter Real tauIntegrator = 0.1; // Integrator time constant
parameter Real tauDifferentiator = 0.1; // Differentiator time constant
parameter Real tauController = 0.1; // Controller time constant
parameter Real tauPlant = 0.1; // Plant time constant
parameter Real tauObserver = 0.1; // Observer time constant
parameter Real tauFilter = 0.1; // Filter time constant
parameter Real tauScheduler = 0.1; // Scheduler time constant
parameter Real tauLogger = 0.1; // Logger time constant
parameter Real tauDebugger = 0.1; // Debugger time constant
parameter Real tauReporter = 0.1; // Reporter time constant
parameter Real tauPrinter = 0.1; // Printer time constant
parameter Real tauWriter = 0.1; // Writer time constant
parameter Real tauReader = 0.1; // Reader time constant
parameter Real tauConnector = 0.1; // Connector time constant
parameter Real tauConnector2 = 0.1; // Connector2 time constant
parameter Real tauConnector3 = 0.1; // Connector3 time constant
parameter Real tauConnector4 = 0.1; // Connector4 time constant
parameter Real tauConnector5 = 0.1; // Connector5 time constant
parameter Real tauConnector6 = 0.1; // Connector6 time constant
parameter Real tauConnector7 = 0.1; // Connector7 time constant
parameter Real tauConnector8 = 0.1; // Connector8 time constant
parameter Real tauConnector9 = 0.1; // Connector9 time constant
parameter Real tauConnector10 = 0.1; // Connector10 time constant
parameter Real tauConnector11 = 0.1; // Connector11 time constant
parameter Real tauConnector12 = 0.1; // Connector12 time constant
parameter Real tauConnector13 = 0.1; // Connector13 time constant
parameter Real tauConnector14 = 0.1; // Connector14 time constant
parameter Real tauConnector15 = 0.1; // Connector15 time constant
parameter Real tauConnector16 = 0.1; // Connector16 time constant
parameter Real tauConnector17 = 0.1; // Connector17 time constant
parameter Real tauConnector18 = 0.1; // Connector18 time constant
parameter Real tauConnector19 = 0.1; // Connector19 time constant
parameter Real tauConnector20 = 0.1; // Connector20 time constant
parameter Real tauConnector21 = 0.1; // Connector21 time constant
parameter Real tauConnector22 = 0.1; // Connector22 time constant
parameter Real tauConnector23 = 0.1; // Connector23 time constant
parameter Real tauConnector24 = 0.1; // Connector24 time constant
parameter Real tauConnector25 = 0.1; // Connector25 time constant
parameter Real tauConnector26 = 0.1; // Connector26 time constant
parameter Real tauConnector27 = 0.1; // Connector27 time constant
parameter Real tauConnector28 = 0.1; // Connector28 time constant
parameter Real tauConnector29 = 0.1; // Connector29 time constant
parameter Real tauConnector30 = 0.1; // Connector30 time constant
parameter Real tauConnector31 = 0.1; // Connector31 time constant
parameter Real tauConnector32 = 0.1; // Connector32 time constant
parameter Real tauConnector33 = 0.1; // Connector33 time constant
parameter Real tauConnector34 = 0.1; // Connector34 time constant
parameter Real tauConnector35 = 0.1; // Connector35 time constant
parameter Real tauConnector36 = 0.1; // Connector36 time constant
parameter Real tauConnector37 = 0.1; // Connector37 time constant
parameter Real tauConnector38 = 0.1; // Connector38 time constant
parameter Real tauConnector39 = 0.1; // Connector39 time constant
parameter Real tauConnector40 = 0.1; // Connector40 time constant
parameter Real tauConnector41 = 0.1; // Connector41 time constant
parameter Real tauConnector42 = 0.1; // Connector42 time constant
parameter Real tauConnector43 = 0.1; // Connector43 time constant
parameter Real tauConnector44 = 0.1; // Connector44 time constant
parameter Real tauConnector45 = 0.1; // Connector45 time constant
parameter Real tauConnector46 = 0.1; // Connector46 time constant
parameter Real tauConnector47 = 0.1; // Connector47 time constant
parameter Real tauConnector48 = 0.1; // Connector48 time constant
parameter Real tauConnector49 = 0.1; // Connector49 time constant
parameter Real tauConnector50 = 0.1; // Connector50 time constant
parameter Real tauConnector51 = 0.1; // Connector51 time constant
parameter Real tauConnector52 = 0.1; // Connector52 time constant
parameter Real tauConnector53 = 0.1; // Connector53 time constant
parameter Real tauConnector54 = 0.1; // Connector54 time constant
parameter Real tauConnector55 = 0.1; // Connector55 time constant
parameter Real tauConnector56 = 0.1; // Connector56 time constant
parameter Real tauConnector57 = 0.1; // Connector57 time constant
parameter Real tauConnector58 = 0.1; // Connector58 time constant
parameter Real tauConnector59 = 0.1; // Connector59 time constant
parameter Real tauConnector60 = 0.1; // Connector60 time constant
parameter Real tauConnector61 = 0.1; // Connector61 time constant
parameter Real tauConnector62 = 0.1; // Connector62 time constant
parameter Real tauConnector63 = 0.1; // Connector63 time constant
parameter Real tauConnector64 = 0.1; // Connector64 time constant
parameter Real tauConnector65 = 0.1; // Connector65 time constant
parameter Real tauConnector66 = 0.1; // Connector66 time constant
parameter Real tauConnector67 = 0.1; // Connector67 time constant
parameter Real tauConnector68 = 0.1; // Connector68 time constant
parameter Real tauConnector69 = 0.1; // Connector69 time constant
parameter Real tauConnector70 = 0.1; // Connector70 time constant
parameter Real tauConnector71 = 0.1; // Connector71 time constant
parameter Real tauConnector72 = 0.1; // Connector72 time constant
parameter Real tauConnector73 = 0.1; // Connector73 time constant
parameter Real tauConnector74 = 0.1; // Connector74 time constant
parameter Real tauConnector75 = 0.1; // Connector75 time constant
parameter Real tauConnector76 = 0.1; // Connector76 time constant
parameter Real tauConnector77 = 0.1; // Connector77 time constant
parameter Real tauConnector78 = 0.1; // Connector78 time constant
parameter Real tauConnector79 = 0.1; // Connector79 time constant
parameter Real tauConnector80 = 0.1; // Connector80 time constant
parameter Real tauConnector81 = 0.1; // Connector81 time constant
parameter Real tauConnector82 = 0.1; // Connector82 time constant
parameter Real tauConnector83 = 0.1; // Connector83 time constant
parameter Real tauConnector84 = 0.1; // Connector84 time constant
parameter Real tauConnector85 = 0.1; // Connector85 time constant
parameter Real tauConnector86 = 0.1; // Connector86 time constant
parameter Real tauConnector87 = 0.1; // Connector87 time constant
parameter Real tauConnector88 = 0.1; // Connector88 time constant
parameter Real tauConnector89 = 0.1; // Connector89 time constant
parameter Real tauConnector90 = 0.1; // Connector90 time constant
parameter Real tauConnector91 = 0.1; // Connector91 time constant
parameter Real tauConnector92 = 0.1; // Connector92 time constant
parameter Real tauConnector93 = 0.1; // Connector93 time constant
parameter Real tauConnector94 = 0.1; // Connector94 time constant
parameter Real tauConnector95 = 0.1; // Connector95 time constant
parameter Real tauConnector96 = 0.1; // Connector96 time constant
parameter Real tauConnector97 = 0.1; // Connector97 time constant
parameter Real tauConnector98 = 0.1; // Connector98 time constant
parameter Real tauConnector99 = 0.1; // Connector99 time constant
parameter Real tauConnector100 = 0.1; // Connector100 time constant
end TestTaskControl;

connect (pOut, pIn);
connect (pOut, p);
connect (p, pOut);
connect (pi, pIn);
connect (pIn, p);
connect (p, pOut);
end TestTaskControl;
```

③ System Simulation with Modelica Tools

# Introduction: vVDR Method

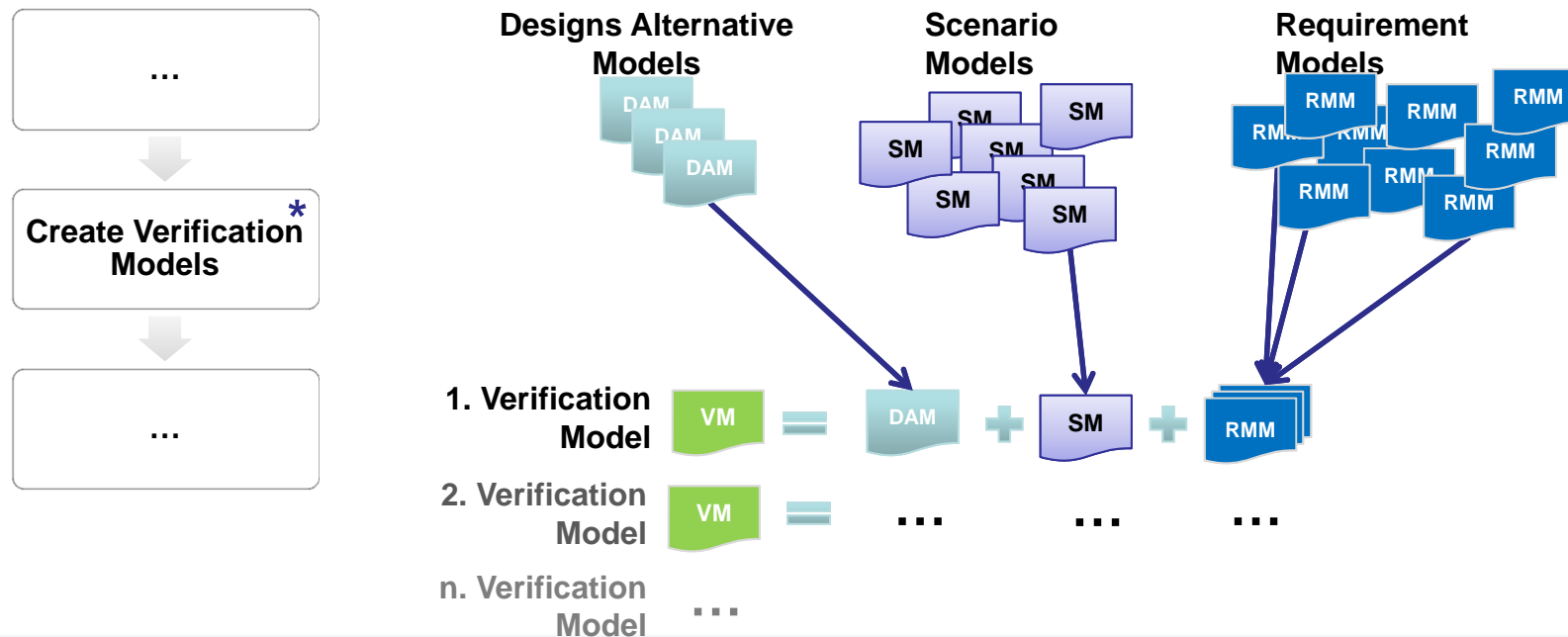


# Challenge

- We want to verify **different design alternatives** against **sets of requirements** using **different scenarios**. Issues:

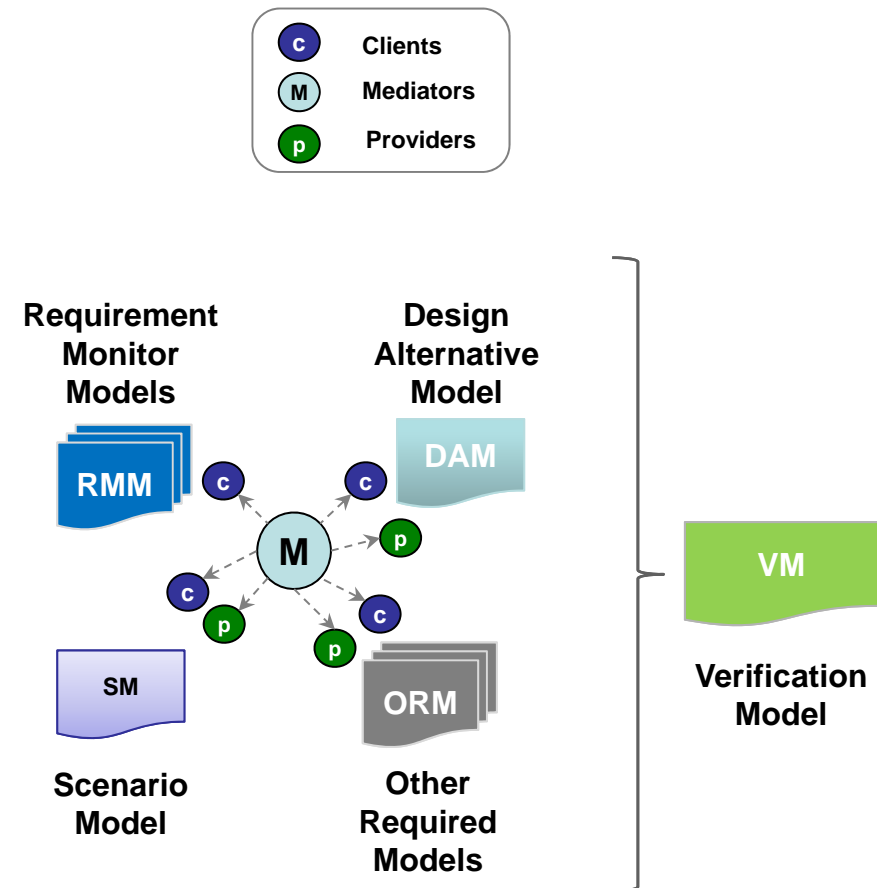
1) How to **find valid combinations** of **design alternatives**, **scenarios** and **requirements** in order to enable an automated composition of verification models?

2) Having found a valid combination: How to **bind all components correctly**?



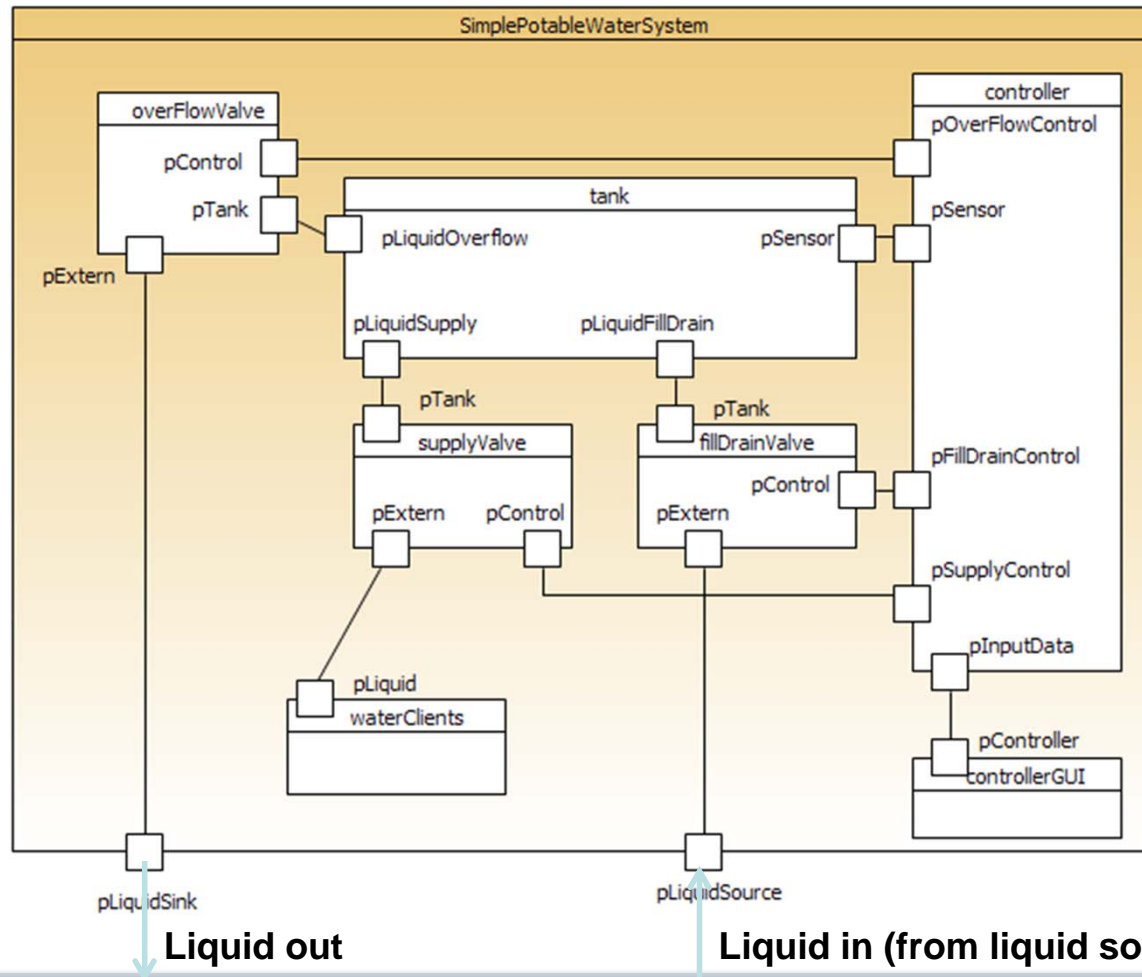
# Solution Proposal: Value Bindings

- **Value Binding** enables the automation of verification model composition
- Value Bindings include the definition of:
  - **Client** (component that requires data from other components)
  - **Provider** (component that provides data for other components)
  - **Mediator** (mediates between clients and providers)
- Depending on which mediators and providers are in place we can:
  - Determine which clients can be satisfied
  - **Find valid combinations** and generate verification models
  - Generate **binding** code for client components in verification models



# Example: Design Alternative Model

- Simplified Aircraft Potable Water System



- Overhead tank system that can be filled using a liquid source from bottom with the aircraft on ground.

- Controller monitors the level of liquid and controls the valves according to its mode (e.g. “fill”-, “drain”-, “pre-selected value fill”-mode).

# Example: Requirement Monitor Model

*”The time to fill an empty tank shall be 300 sec. max.”*

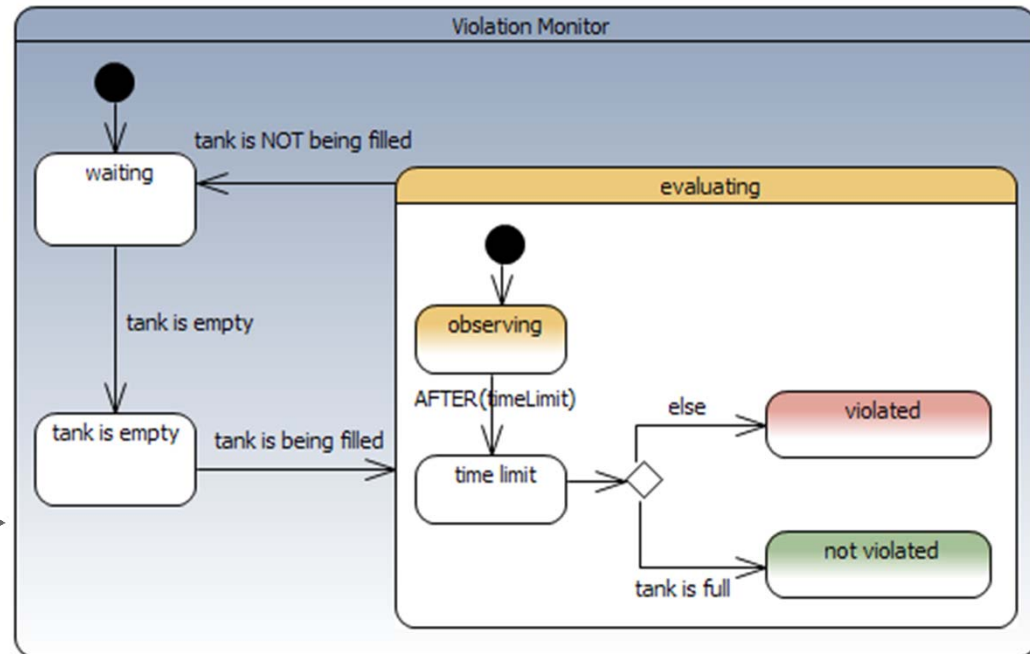
Clients to get input values from design model providers

Rq 001 - Tank filling time

- input Boolean tankIsEmpty = false
- input Boolean tankIsBeingFilled = false
- input Boolean tankIsFull = false
- parameter Real timeLimit = 300
- output Integer status
- Violation Monitor

“status” is set by the violation monitor and indicates the following:

- 0 = not evaluated
- 1 = evaluated and not violated
- 2 = violated

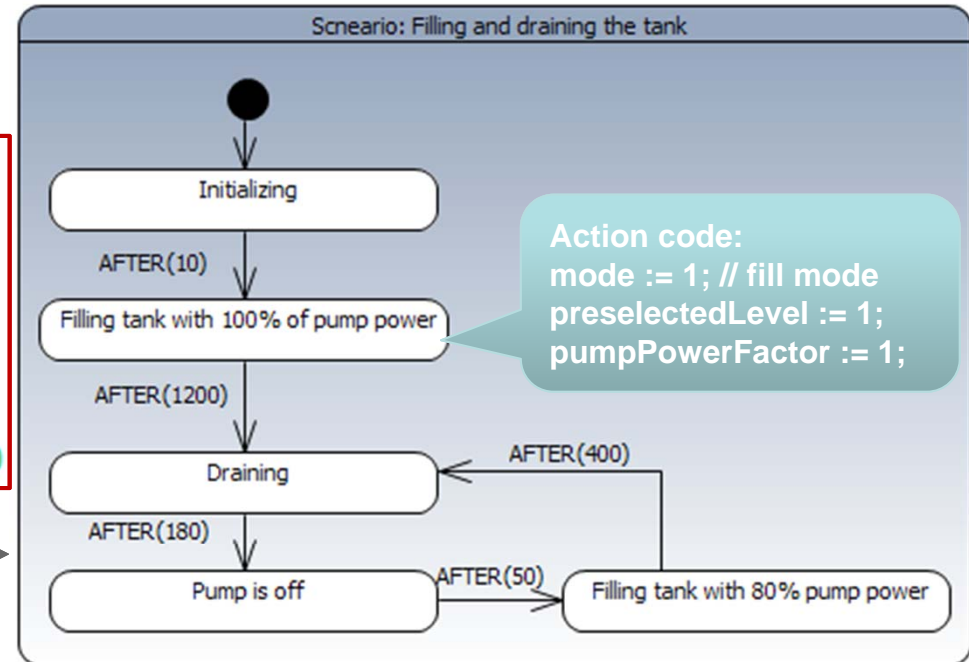




# Example: Scenario Model “Filling and draining the tank”

## Providers for design model clients

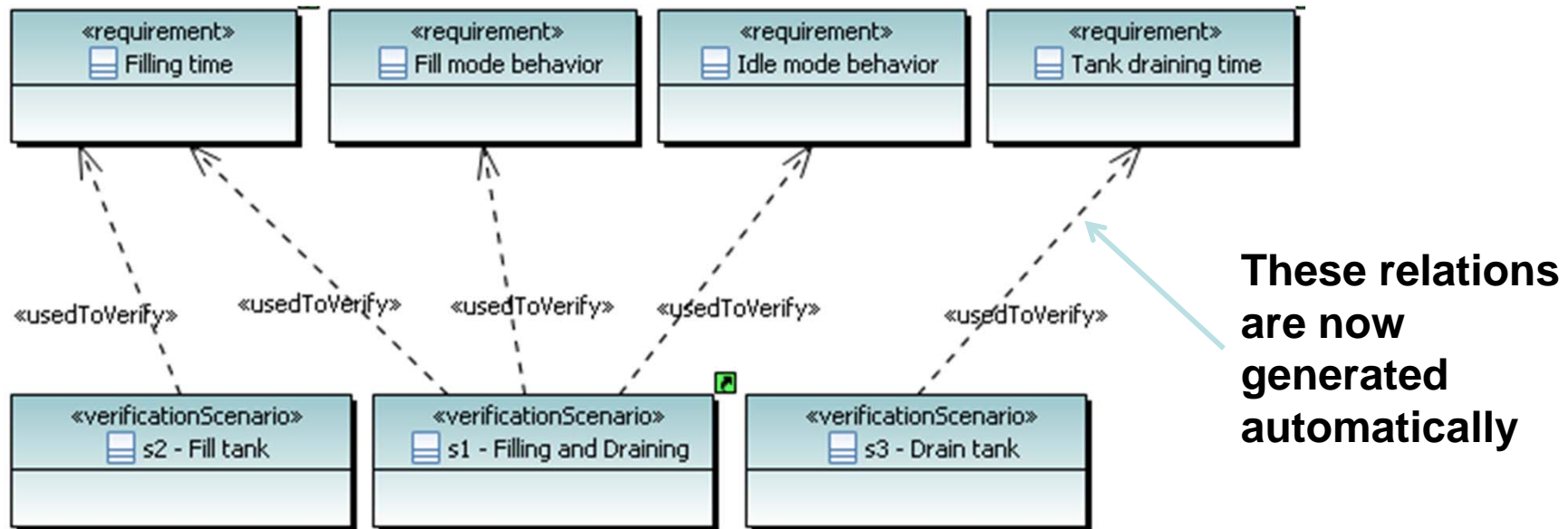
- ▶ `parameter output Real ambPressure(..) = 101325`
- ▶ `output Integer mode(..)`
- ▶ `output Real pumpPowerFactor`
- ▶ `output Real preselectedLevel`
- ▶ `output Integer overflowValveStuckAt = 0`
- ▶ `output Integer fillDrainValveStuckAt = -1`
- ▶ `output Integer supplyVavleStuckAtPosition = 100`
- ▶ `input Real tankHeight = 1`
- ▶ Scenario: Filling and draining the tank



Example scenario: Tank cleaning by filling and draining the tank several times when the aircraft is on ground.

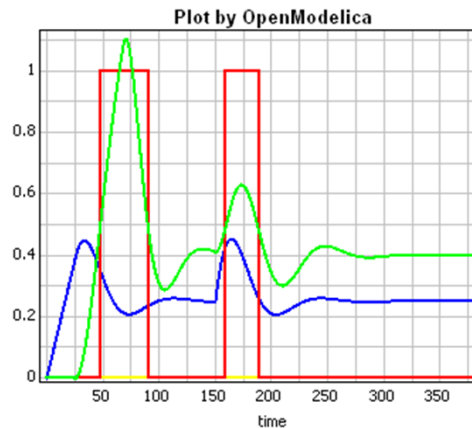
# Example: Mapping Scenarios to Requirements

- Automatic generation/selection of which scenarios are appropriate to verify which requirements
  - **One scenario** can be used to **verify multiple requirements** (to increase requirements coverage and confidence in verification results)
  - Each requirement should be referenced by at least one scenario



# Simulation and Report Generation in ModelicaML

- Verification models are simulated.
- The generated Verification Report is a prepared summary of:
- Configuration, bindings
  - Violations of requirements
  - etc.



Verification models number (3), executed (3), passed (0), failed (3)

**Failed** [VeM for: s1-Fill and Drain Tank \(Plot\)](#)

**Failed** [VeM for: s2-Fill tank \(Plot\)](#)

**Failed** [VeM for: s3-Drain tank \(Plot\)](#)

**Failed** [VeM for: s1-Fill and Drain Tank \(Plot\)](#)

(ModelicaMLModel::GenVeMs for: SPWS Environment\_1::VeM for: s1-Fill and Drain Tank)

**Settings:** startTime = 0, stopTime = 1500, tolerance = default, intervals = 0, outputFormat = plt

verdict [allRequirementsEvaluated](#) : **yes**

verdict [someRequirementsViolated](#) : **yes**

Model to be verified: [SPWS Environment](#)

(ModelicaMLModel::Design::SPWS Environment)

Verification Scenario: [s1-Fill and Drain Tank](#)

(ModelicaMLModel::Verification Scenarios::s1-Fill and Drain Tank)

*mandatory client:* [vs s1 fill and drain tank.tankHeight](#) (changed its value)

Type : = ModelicaReal

Variability : = continuous

Binding code : = sm\_spws\_environment.spws.tank.height

**Violated** Requirement: [Drain mode behavior \(ID 004\)](#)

(ModelicaMLModel::Requirements::Drain mode behavior)

**Text:** When the system is drained only the fill/drain valve should be open, all other valves should be closed.

verdict [evaluated](#) : **yes**

verdict [violated](#) : **yes**

*mandatory client:* [req 004 drain mode behavior.fillDrainValveIsOpen](#) (changed its value)

Type : = ModelicaBoolean

Variability : = continuous

Binding code : = sm\_spws\_environment.spws.fillDrainValve.isFullyOpen

*mandatory client:* [req 004 drain mode behavior.otherValvesAreClosed](#) (changed its value)

Type : = ModelicaBoolean

Variability : = continuous

Binding : = if sm\_spws\_environment.spws.overflowValve.isFullyClosed and sm\_spws\_environment.spws.supplyVavle.isFullyClosed  
code then true else false

# Conclusion

---

- The ModelicaML Value Bindings approach enables automated model composition, which is used in ModelicaML for **automatic generate verification models**
- Bindings do not modify client or provider models (important when libraries are used)
- Using binding definitions we can find **valid combinations** and **automatically generate verification** models
- The generated **verification models** become artifacts that are **created automatically on-demand** and do not need to be maintained

# Overall Summary

---

- Goal of integrated model-based development

This talk covers two aspects

- Integrated static/dynamic debugging of models
  - Dynamic debugging of large algorithmic models fully functional
  - Static Equation debugging prototype need to be integrated and scaled up for large models
- Requirements traceability and verification
  - Automated dynamic verification and generation of verification models
  - Need to be integrated in Modelica standard