

ModelicaML – Tutorial

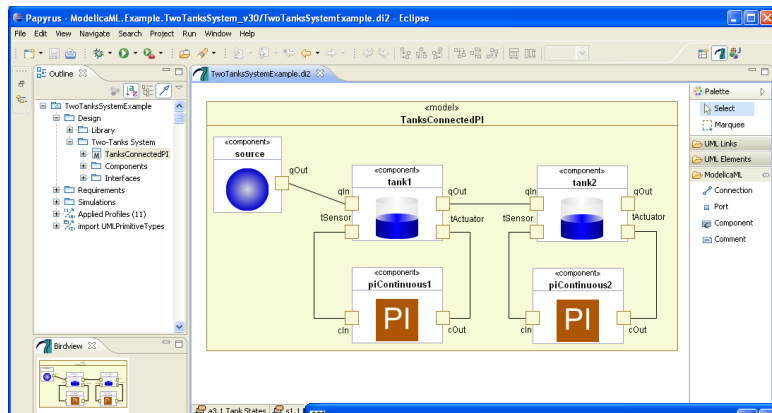
Getting Started

Wladimir Schamai
EADS Innovation Works
Systems Engineering

20.03.2011

ModelicaML: Technology

1 System Modeling with ModelicaML



2 Modelica Code Generation

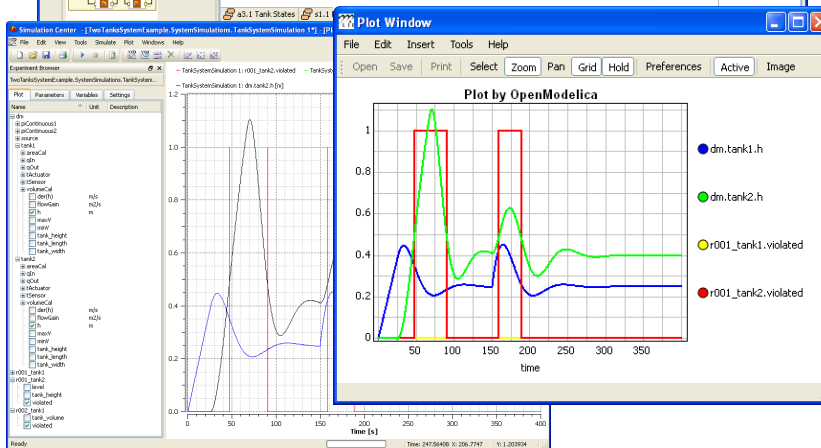


```

1 within TwoTanks;
2
3 function limitValue
4
5   input Real pMin;
6   input Real pMax;
7   output Real pLim;
8   input Real p;
9
10  algorithm
11  // code generated from the Activity "algorithm (diagram)"
12  // Activity.name: "algorithm (diagram)"
13  if p < pMin then
14    pLim := pMin; // OpaqueAction.name: "pLim := pMin;"
15  elseif p > pMax then
16    pLim := pMax;
17  else
18    pLim := p;
19  end if;
20
21 end limitValue;
22
23 model Tank
24
25   TwoTanksSystemExample.DesignModels.LiquidSource source(flowLevel = 0.02);
26   TwoTanksSystemExample.DesignModels.Tank tank1;
27   TwoTanksSystemExample.DesignModels.Tank tank2(tank_length = 1);
28   TwoTanksSystemExample.DesignModels.PIContinuousController piContinuous1(ref = 0.25);
29   TwoTanksSystemExample.DesignModels.PIContinuousController piContinuous2(ref = 0.4);
30
31   parameter Real maxV = 0; //Limits for output valve flow
32   parameter Real minV = 10; //Limits for output valve flow
33   Real h(start = 0.0, unit = "m"); //Tank level
34   parameter Real tank_height = 0.6;
35   parameter Real tank_width = 1;
36   parameter Real tank_length = 1.0;
37
38 end Tank;
39
40 within DesignModels;
41
42 model TanksConnectedPI
43
44   TwoTanksSystemExample.DesignModels.LiquidSource source(flowLevel = 0.02);
45   TwoTanksSystemExample.DesignModels.Tank tank1;
46   TwoTanksSystemExample.DesignModels.Tank tank2(tank_length = 1);
47   TwoTanksSystemExample.DesignModels.PIContinuousController piContinuous1(ref = 0.25);
48   TwoTanksSystemExample.DesignModels.PIContinuousController piContinuous2(ref = 0.4);
49
50   connect(source.qOut, tank1.qIn);
51   connect(piContinuous1.cIn, tank1.tSensor);
52   connect(tank1.qOut, tank2.qIn);
53   connect(piContinuous2.cIn, tank2.tSensor);
54   connect(piContinuous2.cOut, tank2.tActuator);
55
56 end TanksConnectedPI;
    
```



3 System Simulation with Modelica Tools

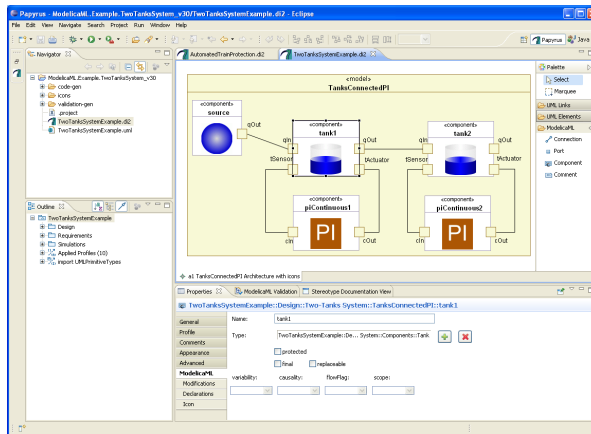


ModelicaML: Technology

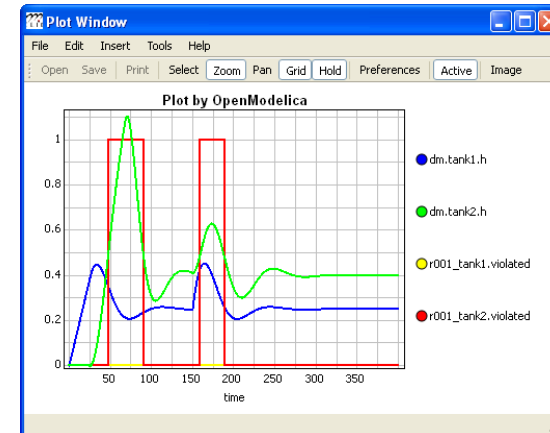


Papyrus UML

ModelicaML Profile (Eclipse Plug-In)



Any Modelica Simulation Tool



Acceleo

ModelicaML Code Generator (Eclipse Plug-In)



Model to Text Transformation



Generated Modelica Code (.mo files)

Required Installations

- If not installed yet: JavaSE-1.6 from <http://www.oracle.com/technetwork/java/javase/overview/index.html>, required for Eclipse.
- Eclipse 3.6 (Helios) Modeling from <http://www.eclipse.org/downloads/>
- Papyrus MDT Eclipse plug-in (find the update site on <http://www.eclipse.org/modeling/mdt/papyrus/updates/index.php>) for modeling with UML
- Acceleo Eclipse plug-in from <http://www.acceleo.org/pages/download-acceleo/en> for Modelica code generation from ModelicaML models
- ModelicaML Eclipse plug-ins from <http://www.openmodelica.org/index.php/developer/tools/134> for modeling with ModelicaML (ModelicaML plug-ins that are available for Papyrus MDT)
- OMC from <http://www.openmodelica.org/index.php/download>. OpenModelica Compiler used for simulation.
- Modelica Development Tooling (MDT) Eclipse plug-in (find the update site on <http://www.openmodelica.org/index.php/developer/tools/133>) for viewing the generated Modelica code if needed

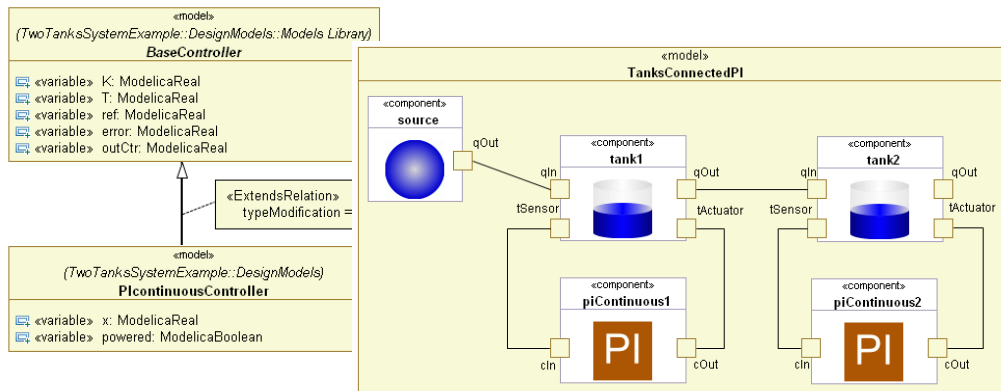
For details see: www.openmodelica.org/modelicaml

ModelicaML

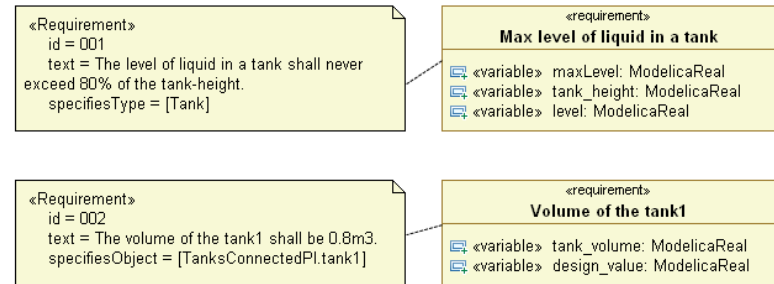
General Description of the
UML-Based Graphical Notation

ModelicaML: Graphical Notation

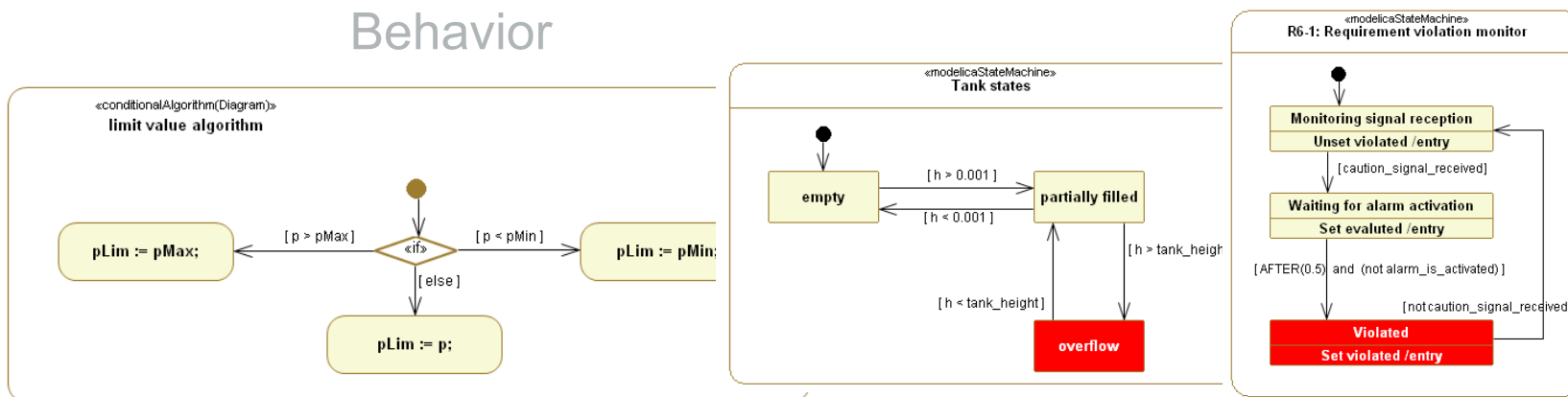
Structure



Requirements



Behavior

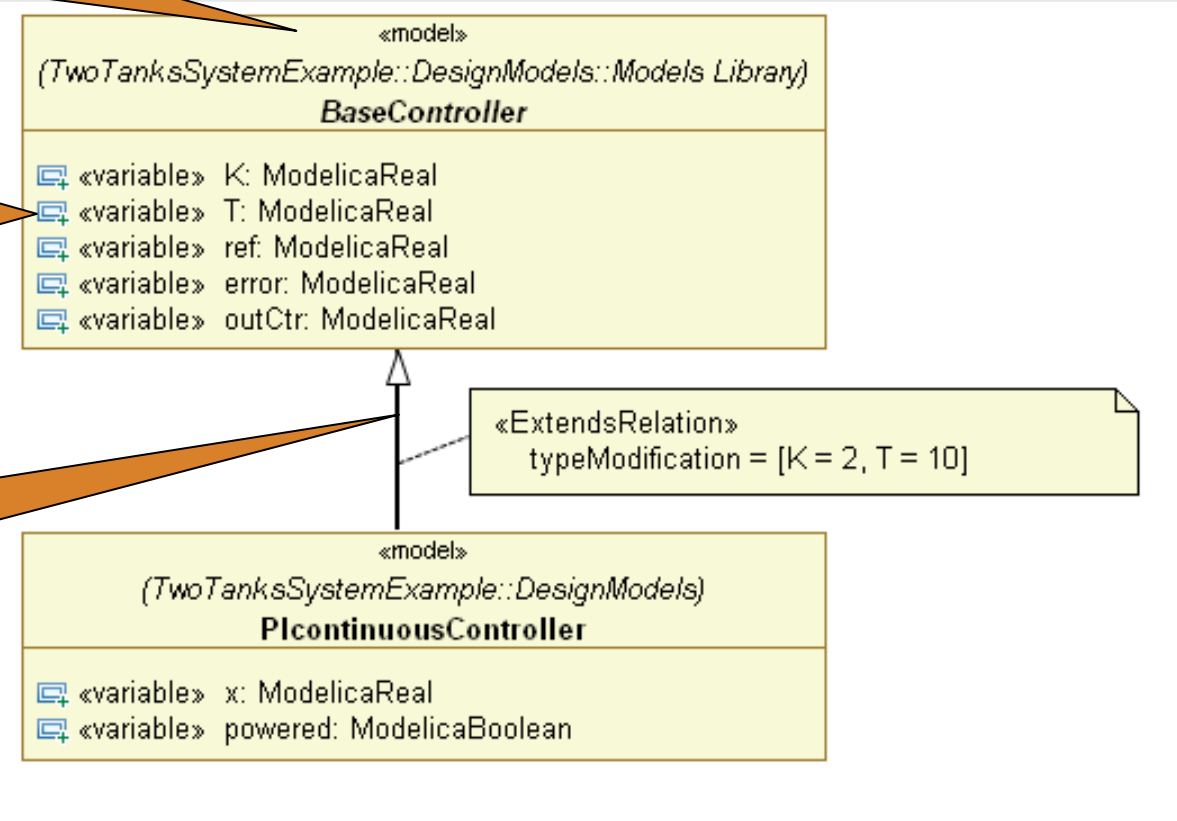


ModelicaML: Class Diagram

Class (model, block, record, connector)

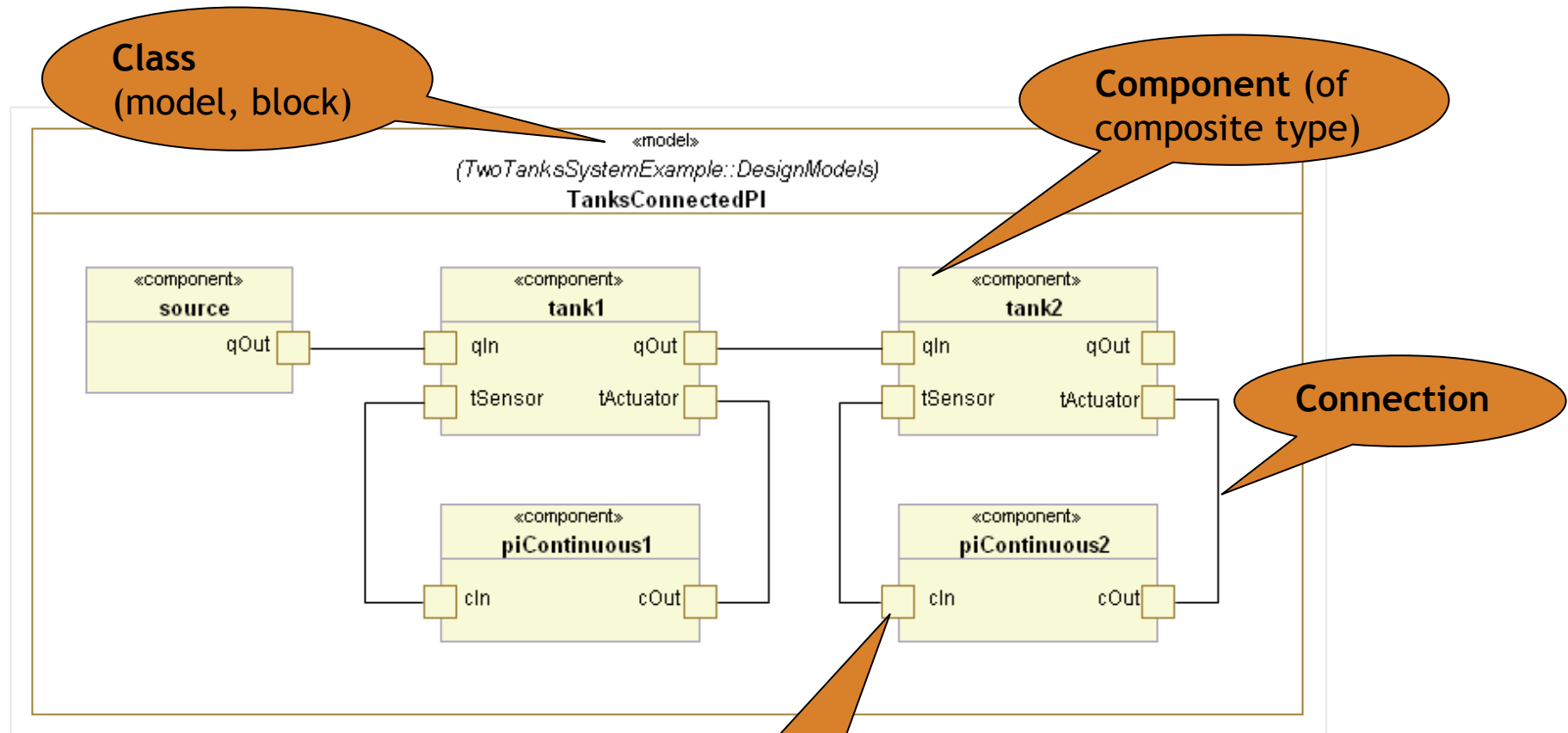
variables (of primitive type) or components (of composite type)

extends relation (with type modifications)



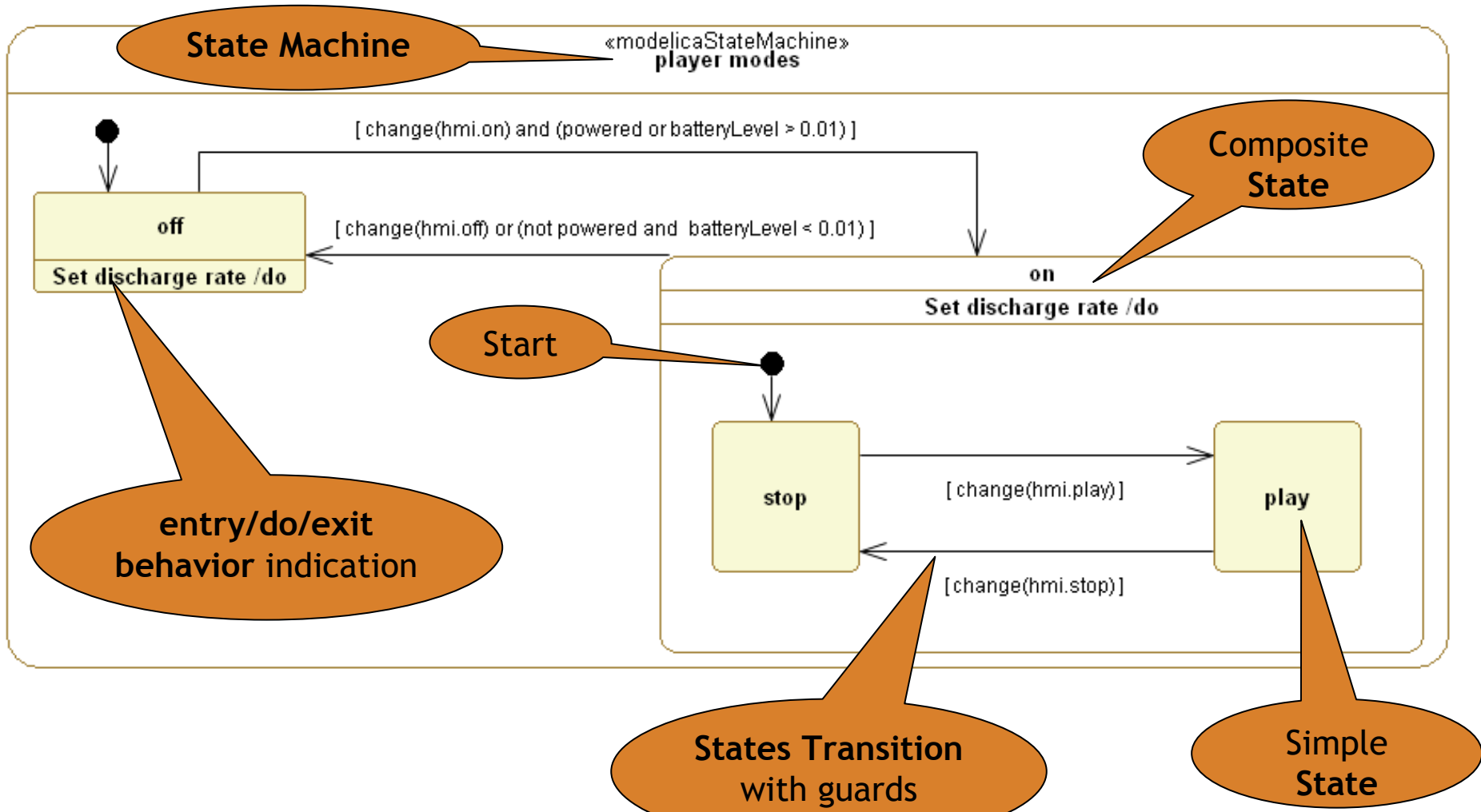
(Two-Tanks System Example)

ModelicaML: Connection Diagram



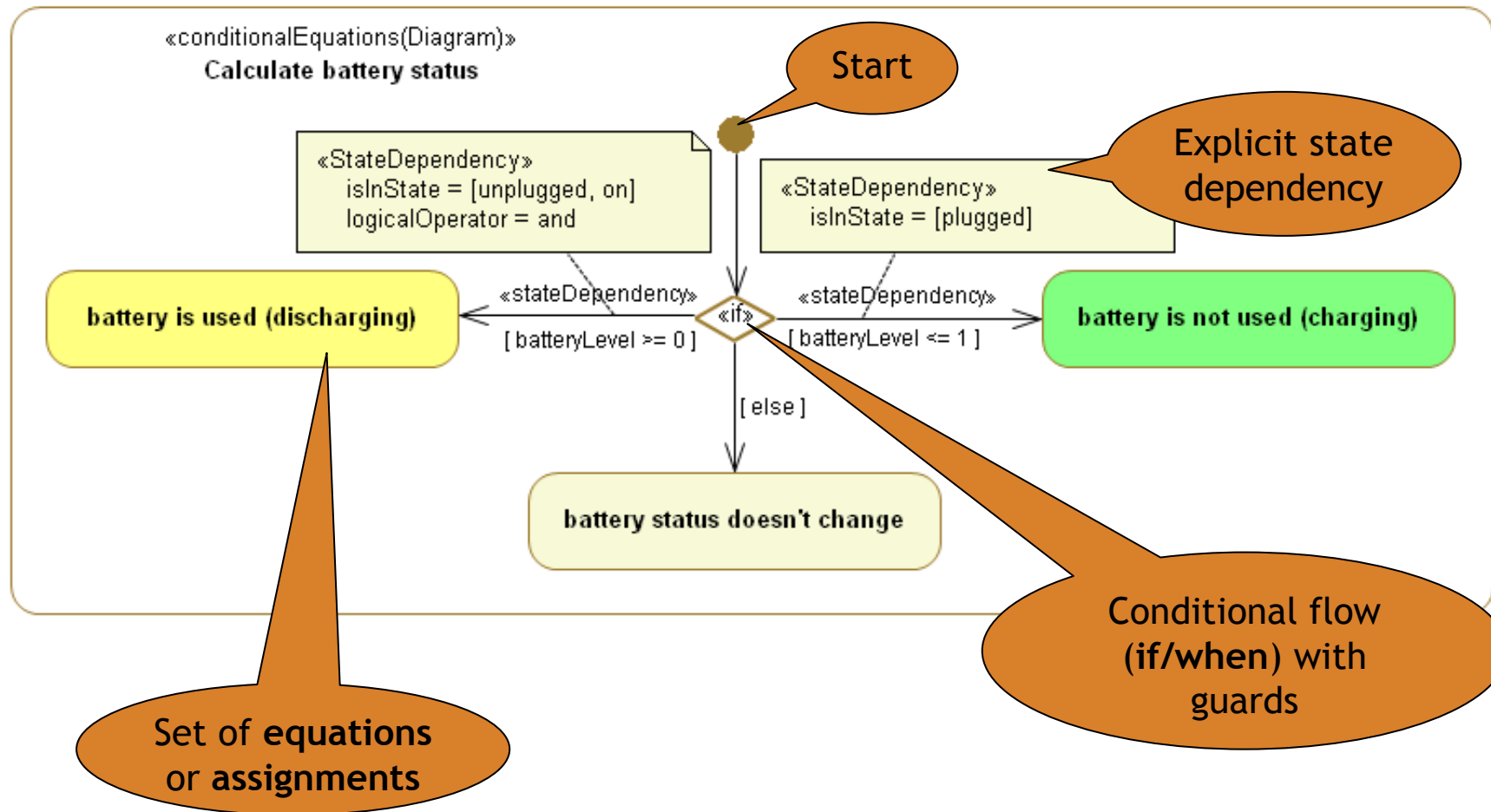
(Two-Tanks System Example)

ModelicaML: State Machine Diagram



(Two-Tanks System Example)

ModelicaML: Conditional Eq./Alg. Diagram



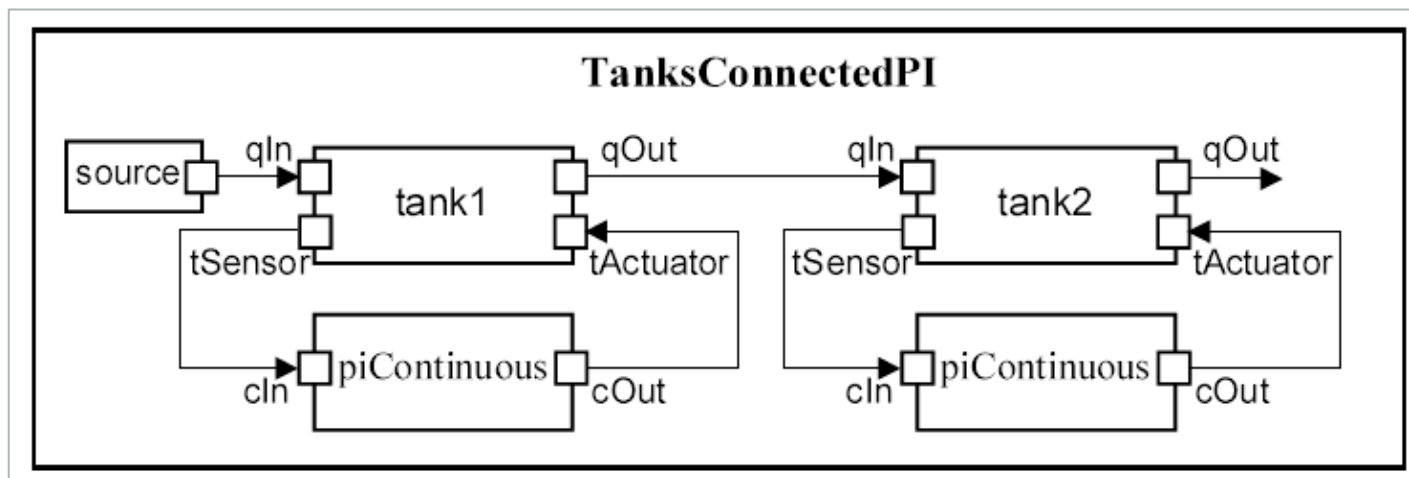
(Player Example)

ModelicaML

Hands-on Modeling Tutorial

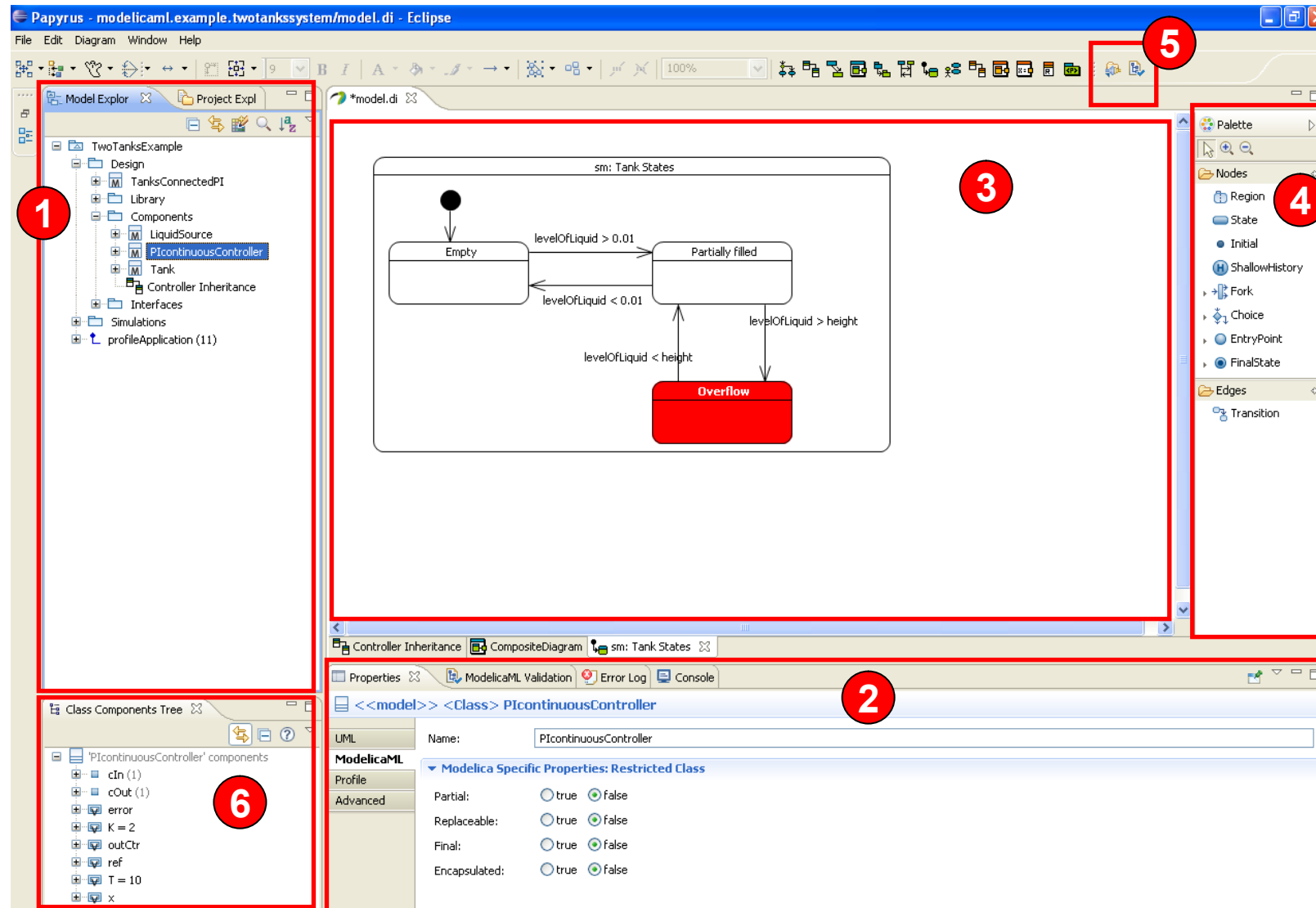
Example: Two Tanks System

- From “source” liquid flows into the “tank1”
- Controller “piContinuous1” controls the level of liquid in “tank1”, based on a predefined reference value, by opening and closing the tank outflow valve.
- Liquid flows from “tank1” into “tank2”
- Controller “piContinuous2” controls the level of liquid in “tank2”



Source: Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Press, 2004. page 391.

ModelicaML Papyrus MDT GUI Overview



The screenshot displays the Papyrus MDT GUI with several key components highlighted by red boxes and numbered 1 through 6:

- 1**: Model Explorer showing the project structure, including the `PIcontinuousController` package.
- 2**: Properties view for the `PIcontinuousController` class, showing ModelicaML specific properties like `Partial`, `Replaceable`, `Final`, and `Encapsulated`.
- 3**: Diagram Editor showing the state machine diagram for `sm: Tank States`, including states like `Empty`, `Partially filled`, and `Overflow`.
- 4**: Palette showing the available nodes and edges for the diagram, such as `Region`, `State`, and `Transition`.
- 5**: The top toolbar containing various tool icons for editing and navigation.
- 6**: Class Components Tree showing the internal components of the `PIcontinuousController`, such as `cIn`, `cOut`, and `error`.

ModelicaML Papyrus MDT GUI Overview

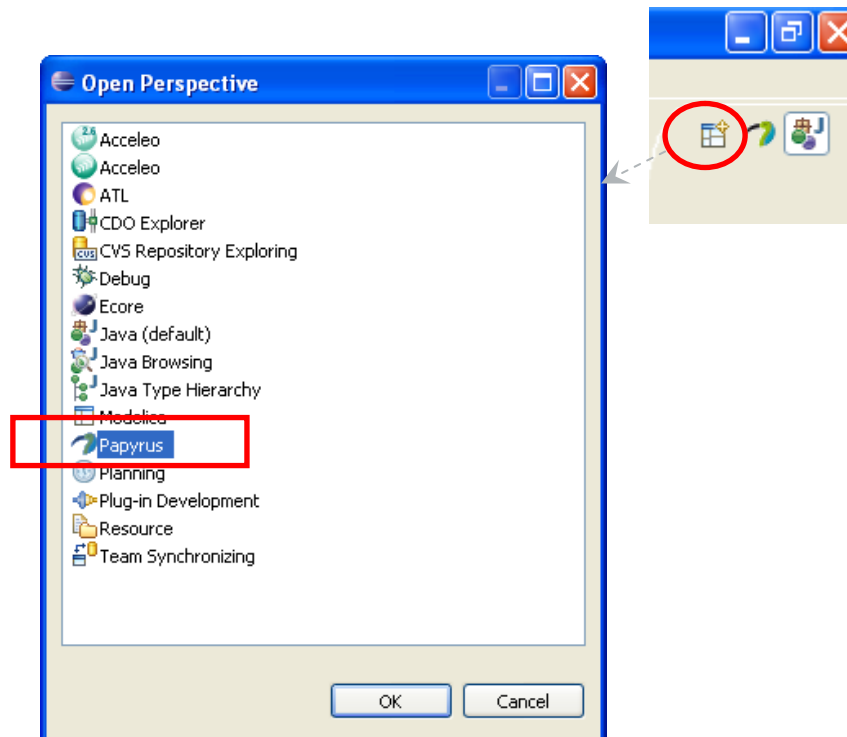
- 1 Model Browser: Shows model elements
- 2 Properties View: Shows the properties of selected element
- 3 Diagram Editors (different UML-based diagrams)
- 4 Palette (different for each diagram)
- 5 ModelicaML code generation and validation buttons
- 6 Component tree: Shows the components hierarchy of the selected class

ModelicaML Project Setup

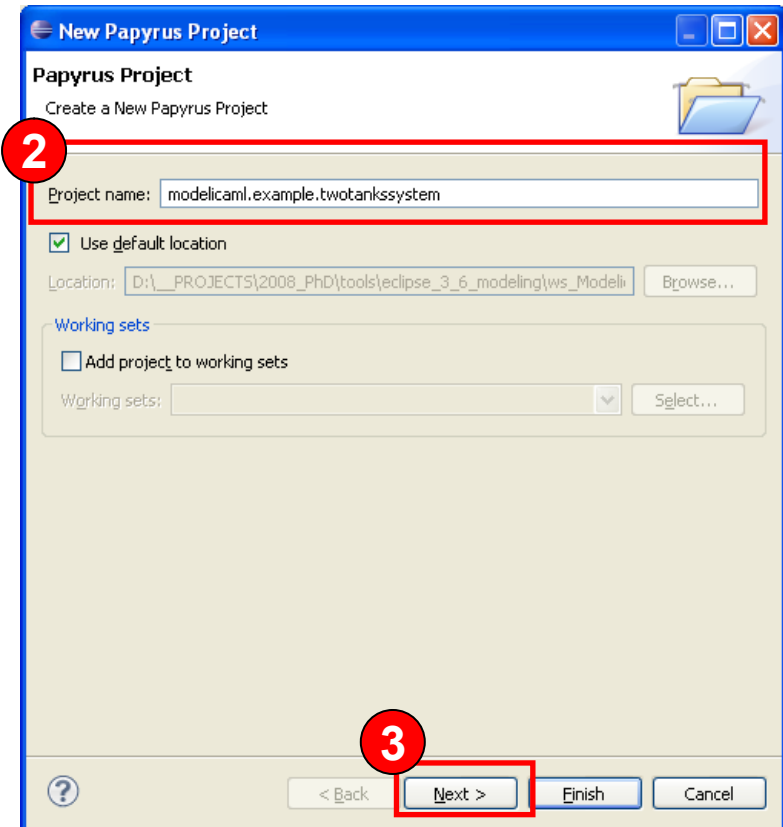
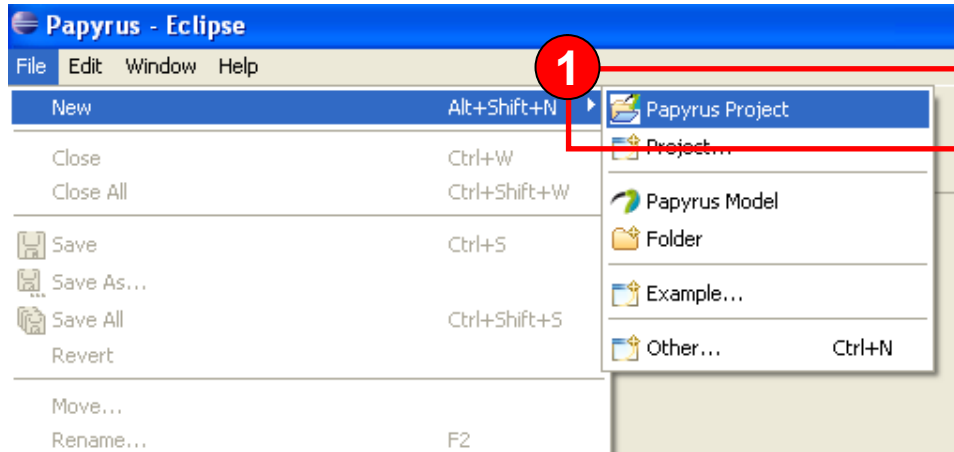


Create ModelicaML Project

- Open Eclipse
- Change the Perspective to Papyrus Perspective

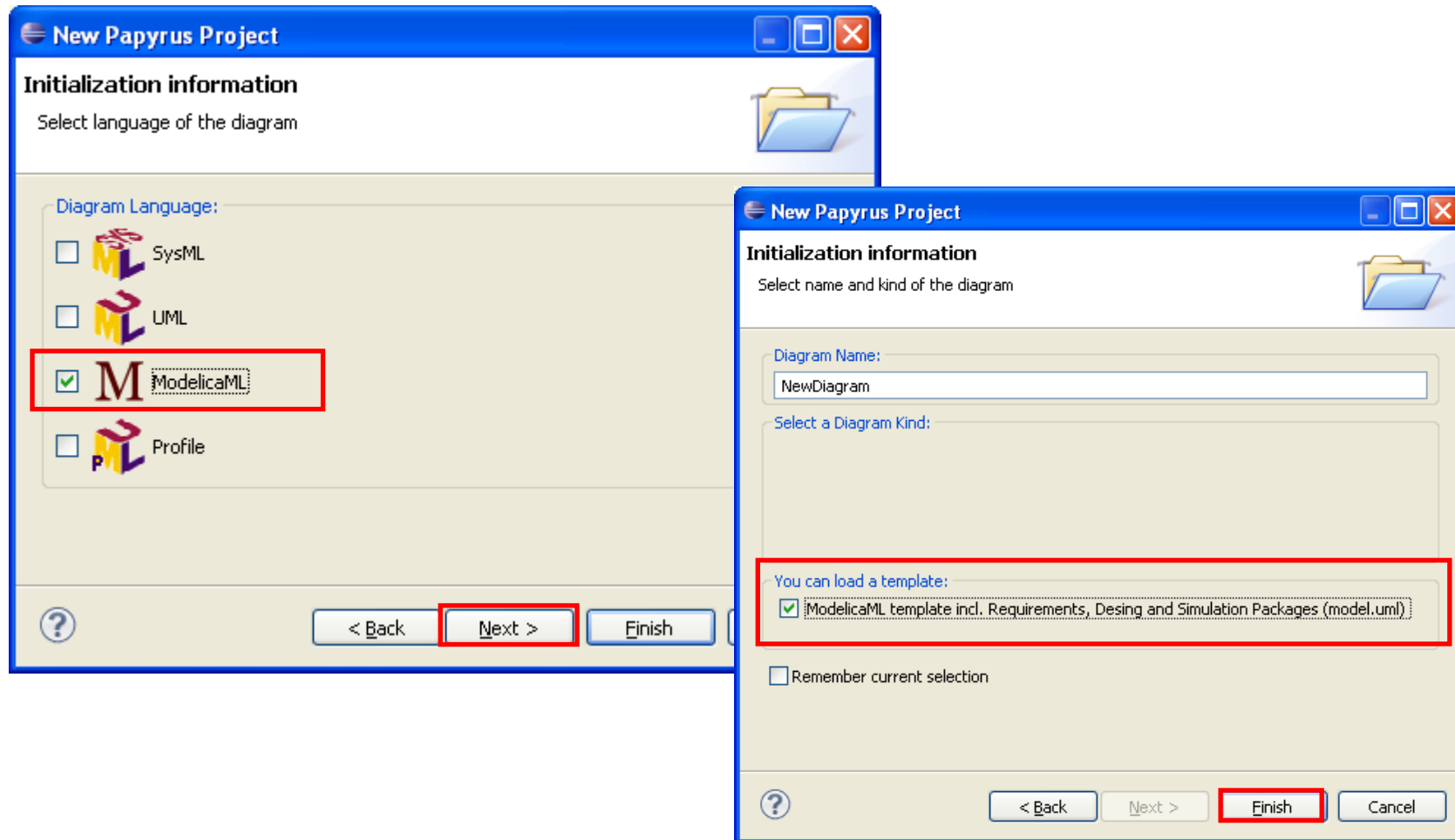


Create ModelicaML Project



Go to File -> Create ...

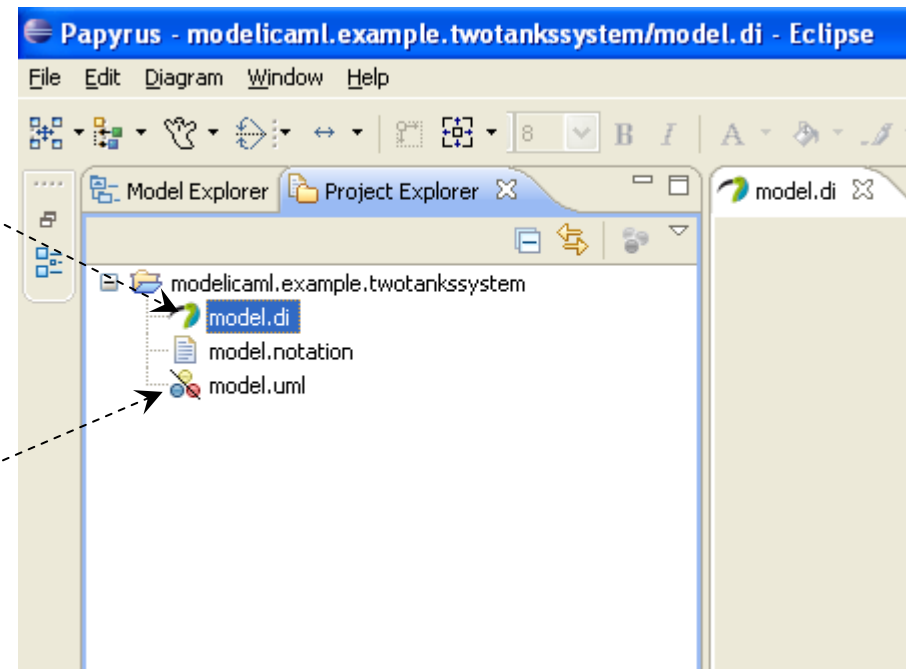
Create ModelicaML Project



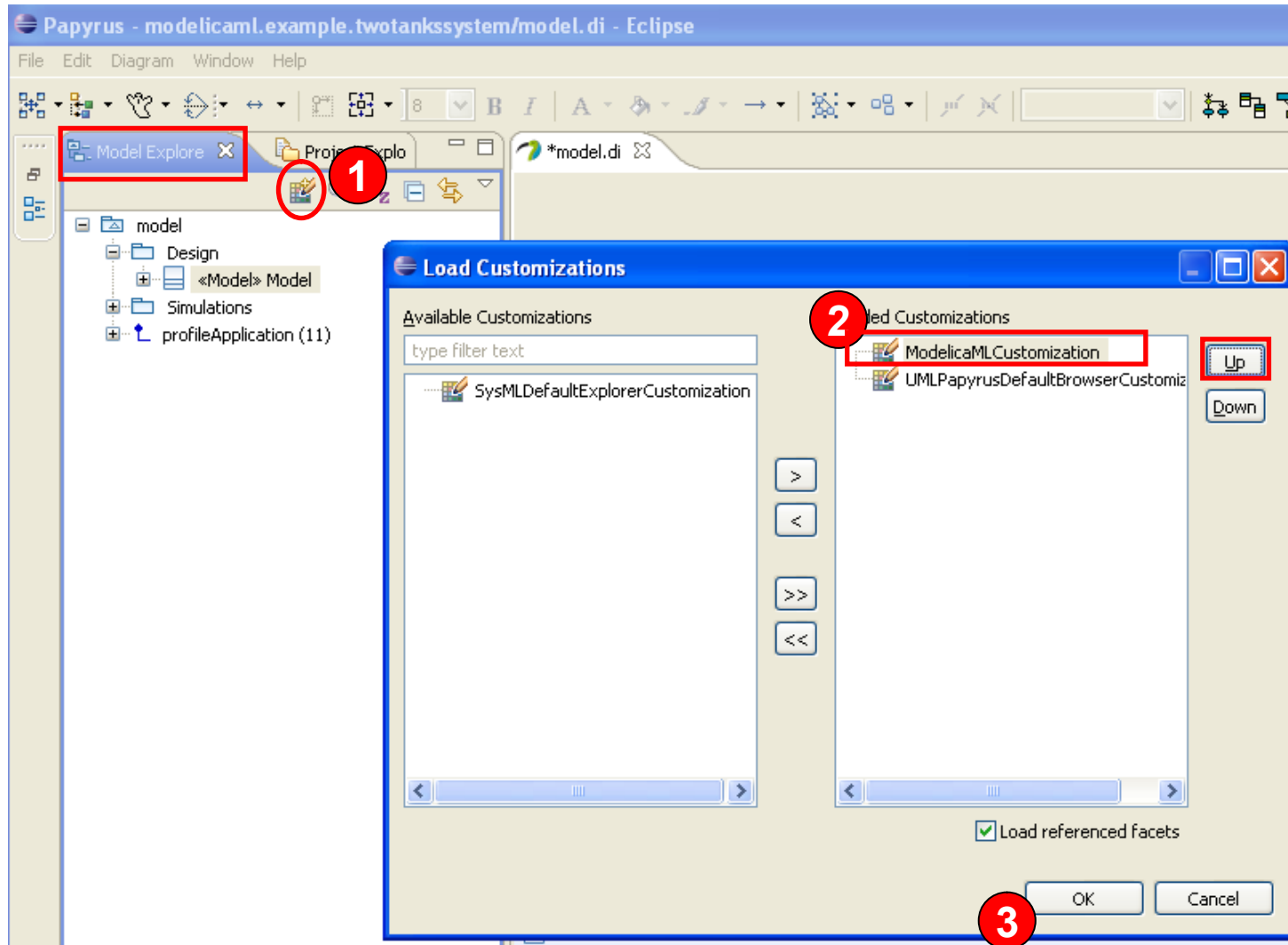
Papyrus Project Files

Diagram file
(can only be edited using Papyrus MDT)

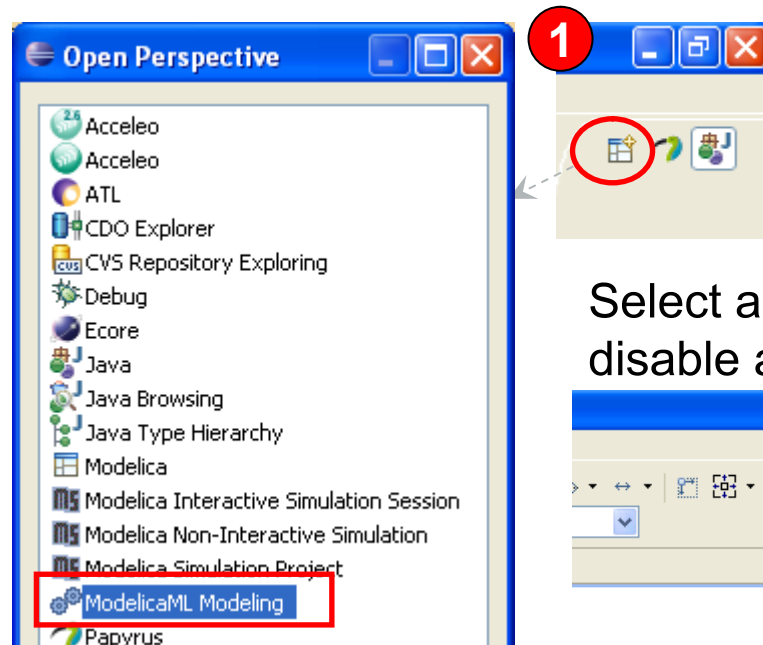
Model file
(can be edited using any UML2 tool)



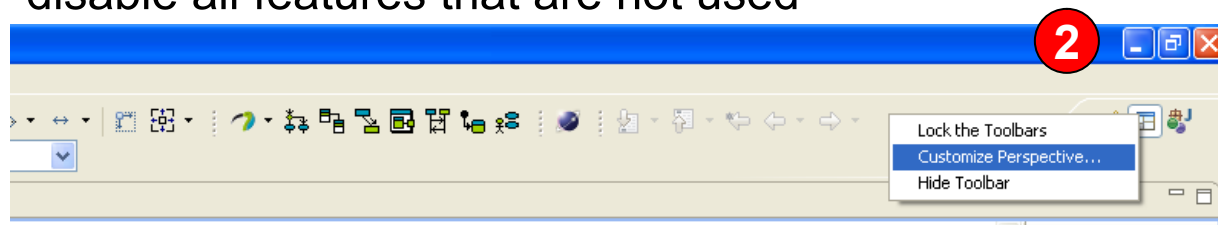
Configure Model Explorer



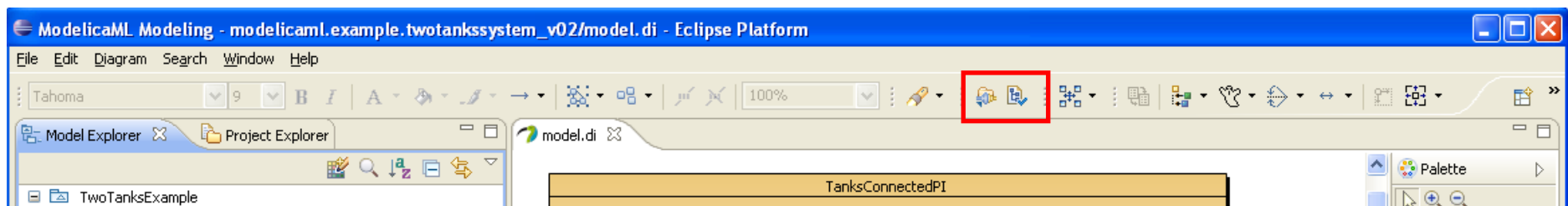
Change and Customize the Perspective



Select and customize the ModelicaML perspective – disable all features that are not used



Typical ModelicaML customized perspective:

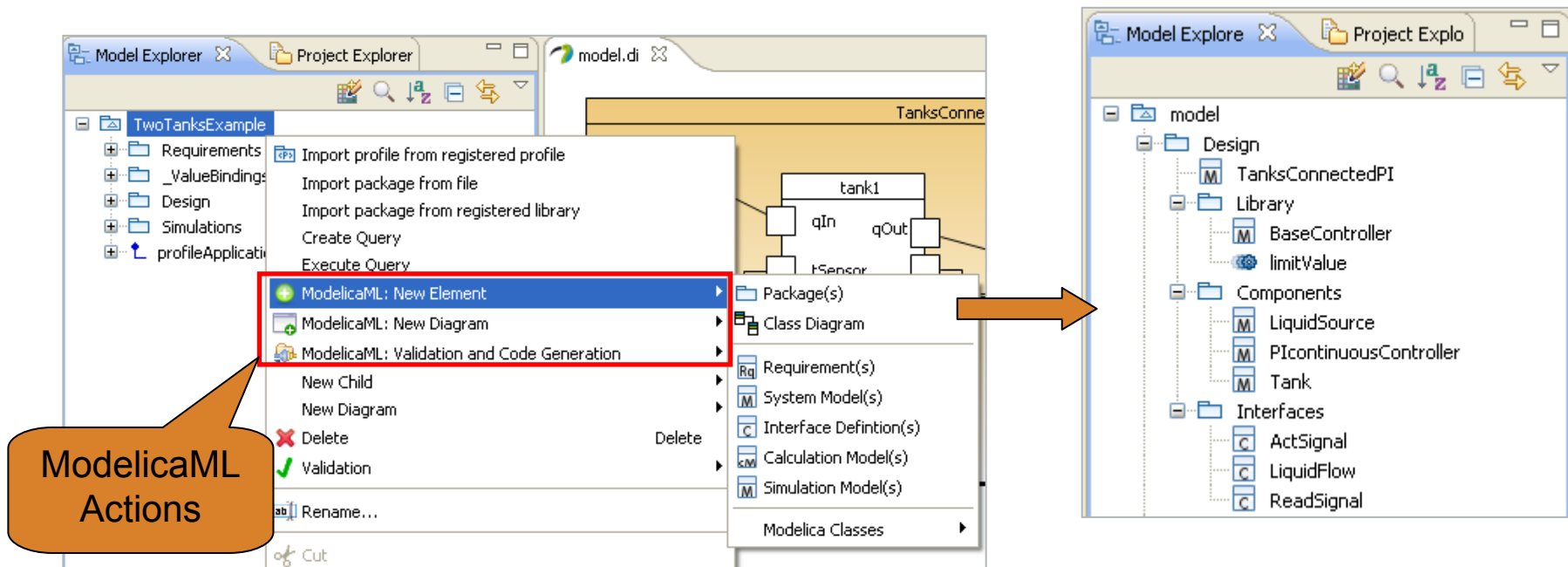


Model Setup

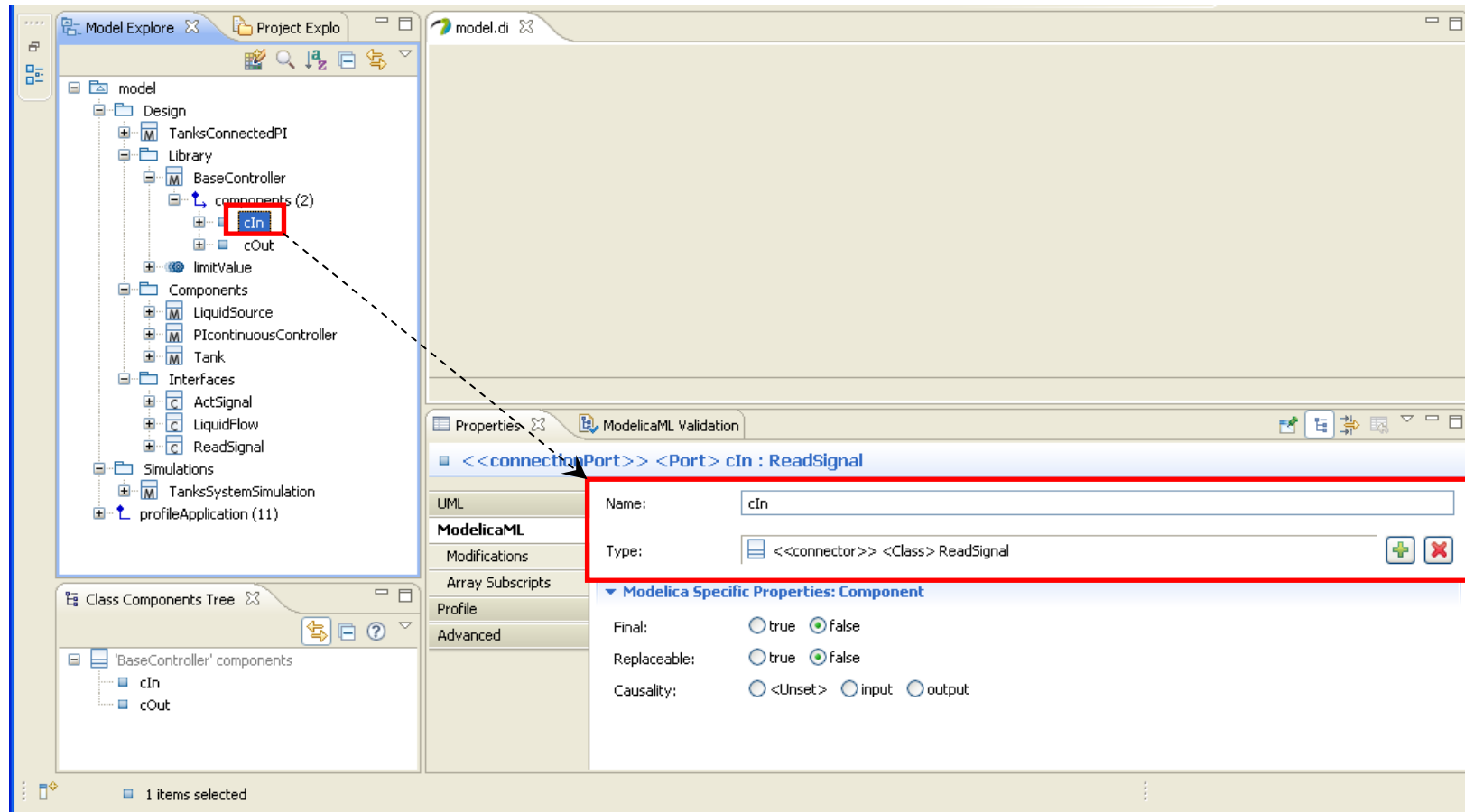


Create Model Structure

Create Packages and Classes using ModelicaML menus



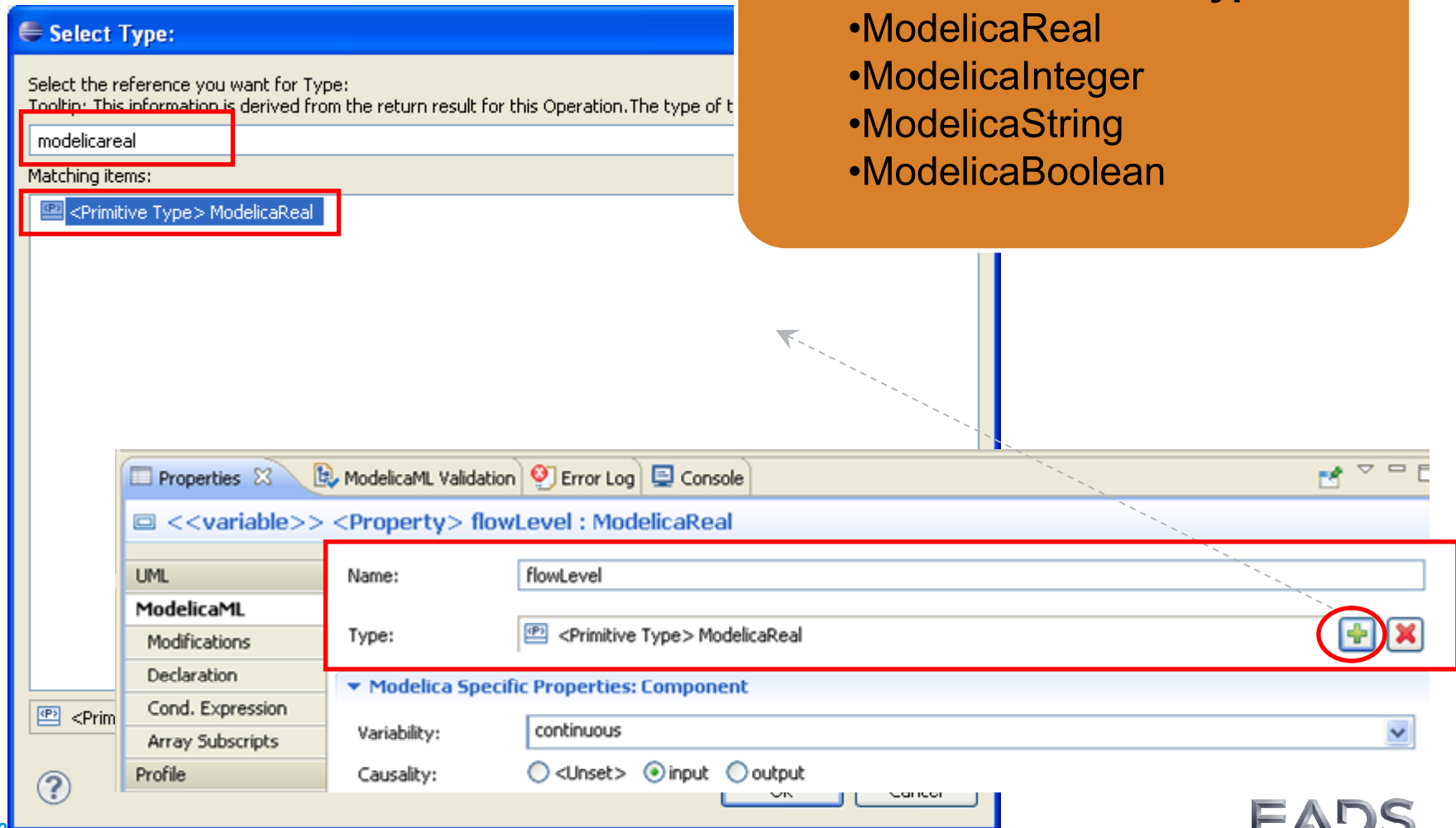
Hint: Setting type of components



Hint: Setting type of components

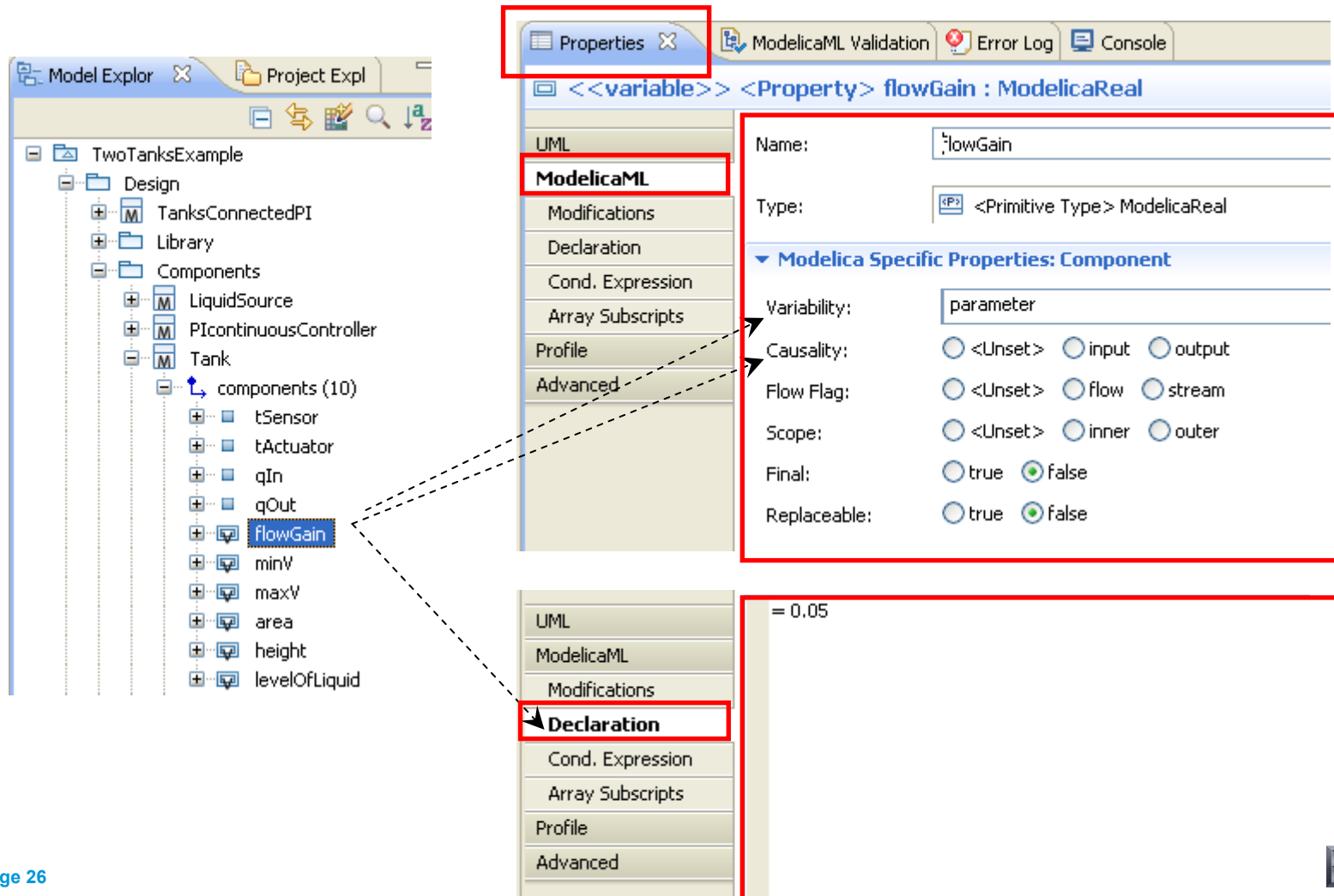
ModelicaML Primitive Types are:

- ModelicaReal
- ModelicaInteger
- ModelicaString
- ModelicaBoolean



The screenshot illustrates the process of setting the type of a component property. In the 'Select Type' dialog, the search term 'modelicareal' is entered, and the matching item '<Primitive Type> ModelicaReal' is selected. In the 'Properties' window, the 'flowLevel' property is set to '<Primitive Type> ModelicaReal'. A red box highlights the type field in both windows, and a dashed arrow points from the type field in the 'Properties' window to the 'Select Type' dialog.

Hint: Setting of the component properties (Declaration, Causality, Variability, etc.)



The screenshot illustrates the configuration of the 'flowGain' property in the ModelicaML IDE. On the left, the 'Model Explorer' shows a project named 'TwoTanksExample' with a 'Tank' component. The 'flowGain' property is highlighted in the 'components (10)' list. On the right, the 'Properties' window is open for the selected property, showing the following configuration:

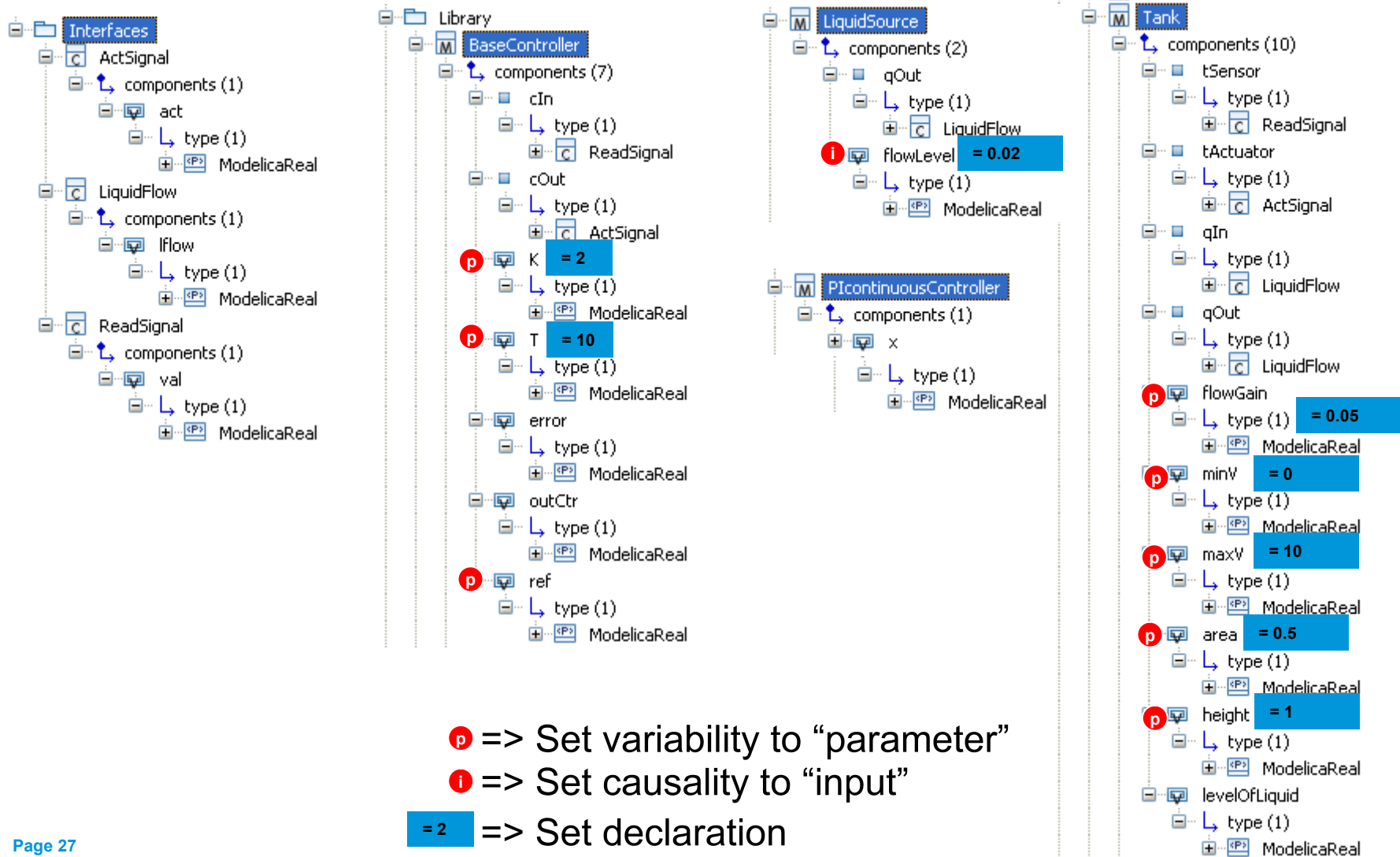
- UML** (highlighted in red)
- ModelicaML** (highlighted in red)
- Modifications**
- Declaration** (highlighted in red)
- Cond. Expression**
- Array Subscripts**
- Profile**
- Advanced**

The 'Modelica Specific Properties: Component' section is expanded, showing the following settings:

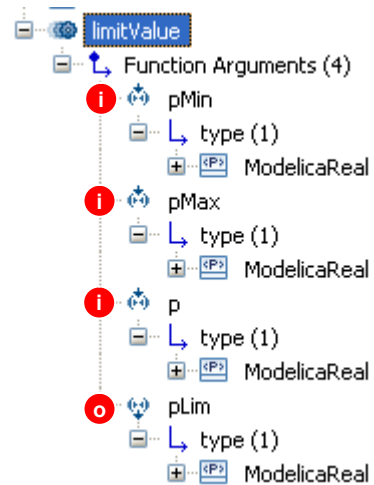
- Name: flowGain
- Type: <Primitive Type> ModelicaReal
- Variability: parameter
- Causality: <Unset> input output
- Flow Flag: <Unset> flow stream
- Scope: <Unset> inner outer
- Final: true false
- Replaceable: true false

The 'Declaration' tab is selected, showing the value: = 0.05

Create Class Components



Create Function Arguments

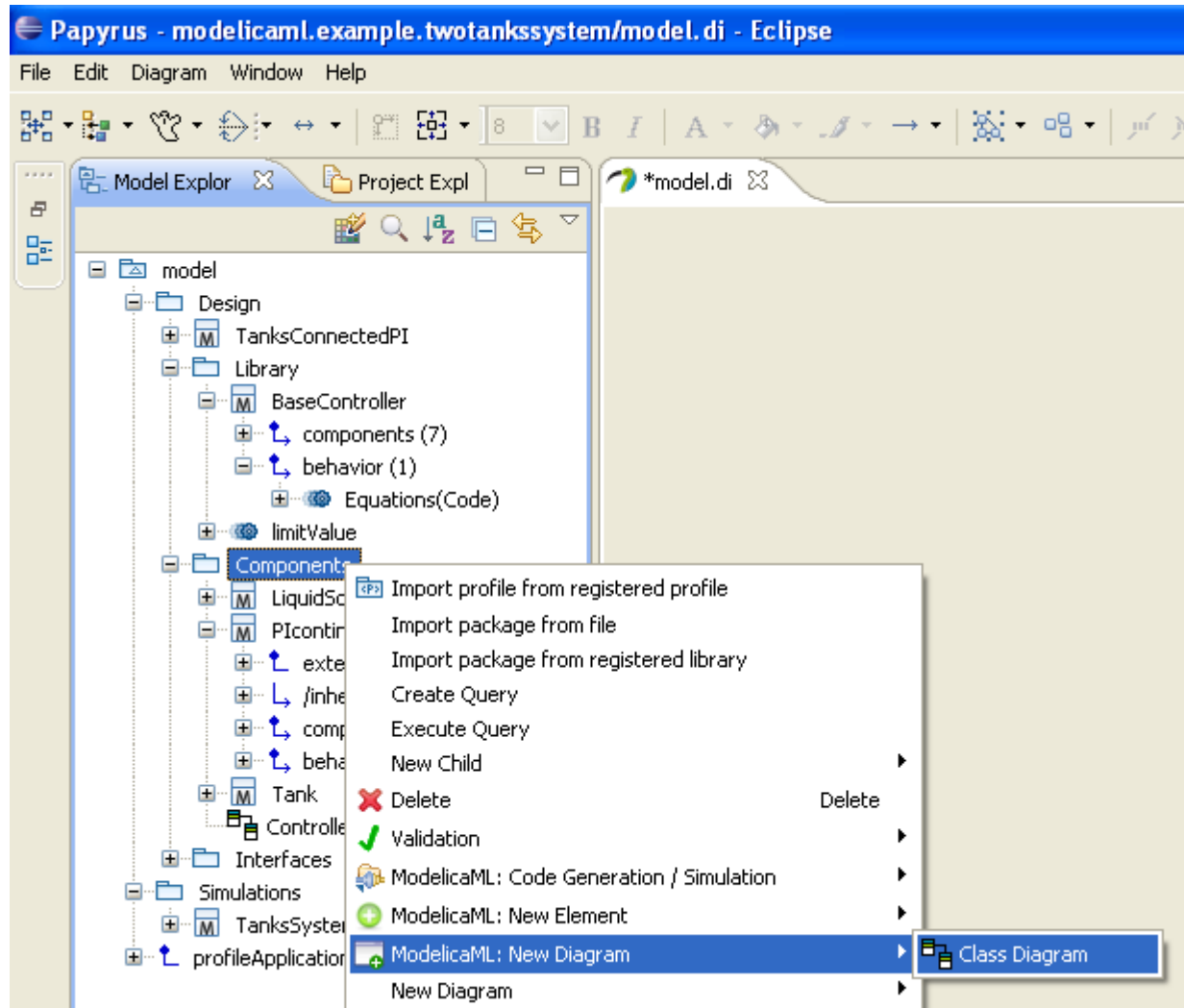


i => Set causality to “input”

o => Set causality to “output”

Inheritance/Extension Modeling

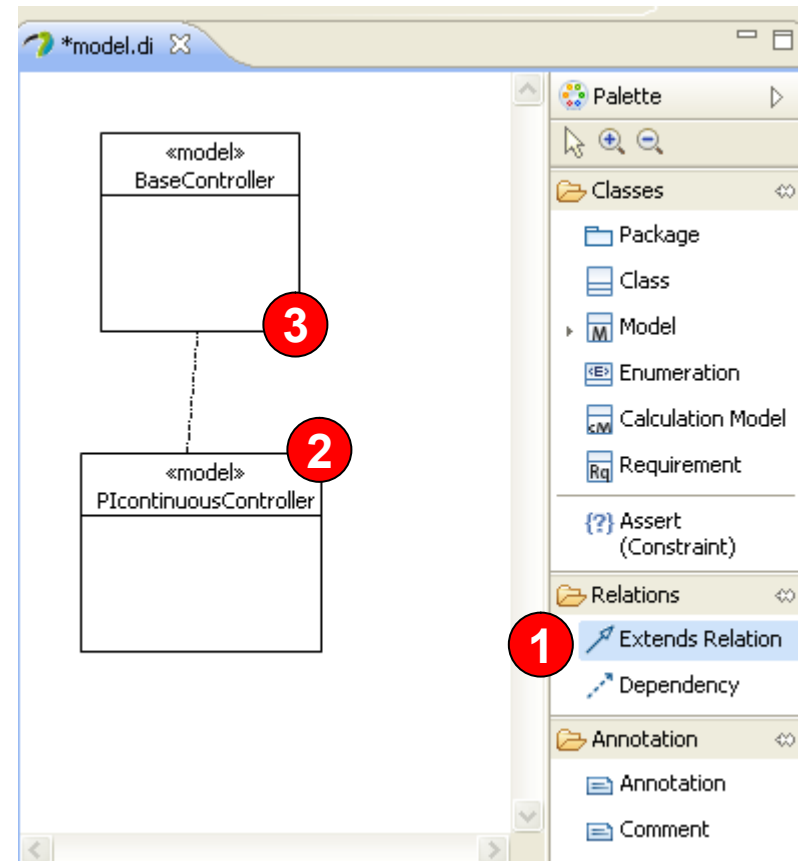
Create Class Diagram



General: Working with diagrams

Creating edges:

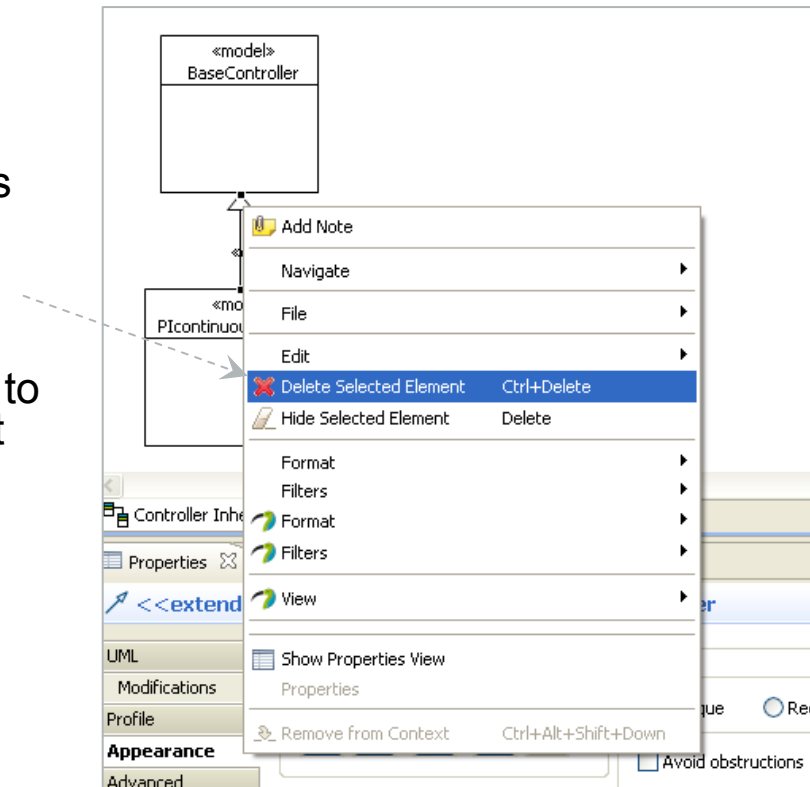
- 1 Select the palette tool
- 2 Click on the edge source element and hold the mouse button
- 3 Move the mouse to the target element and release the mouse button



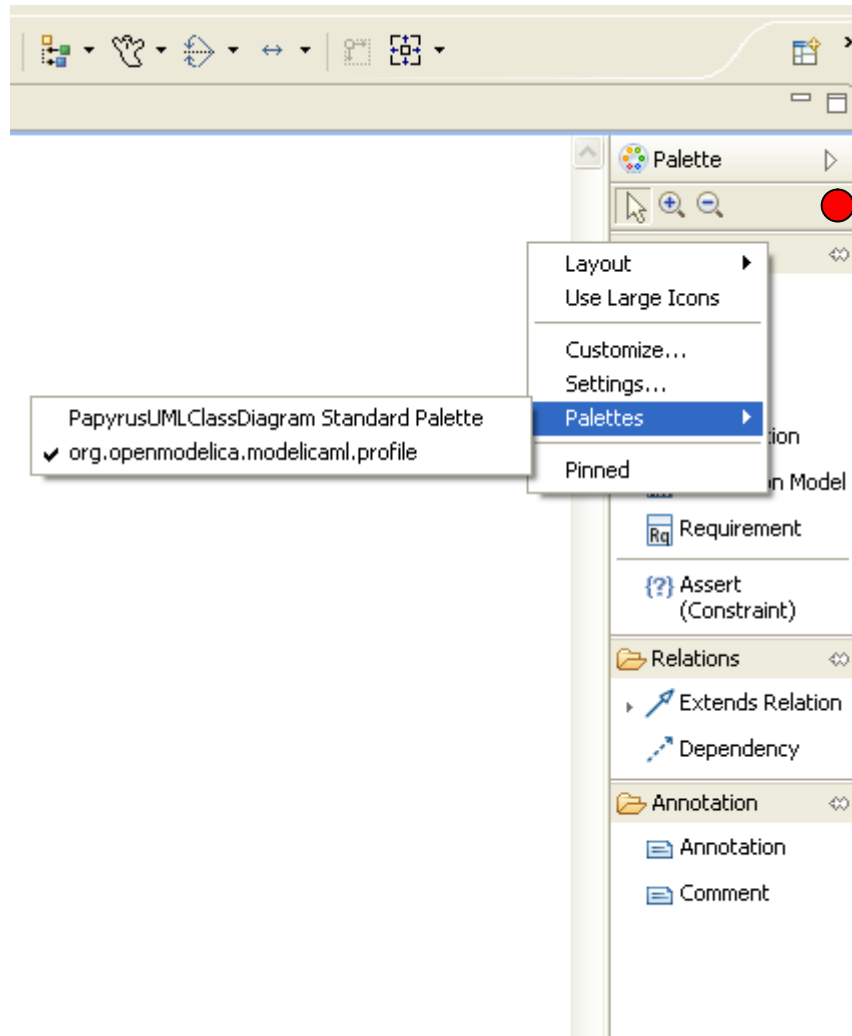
General: Working with diagrams

Deletion of elements:

- Right click on a diagram element
- (preferred) Select the option “Delete Selected Element” in order to delete it from the model. This is recommended in order to keep the model and the diagram consistent
- Select the option “Hide Selected Element” in order to remove the element from the diagram. The element will still exist in the model and can be shown on the diagram by drag&drop.



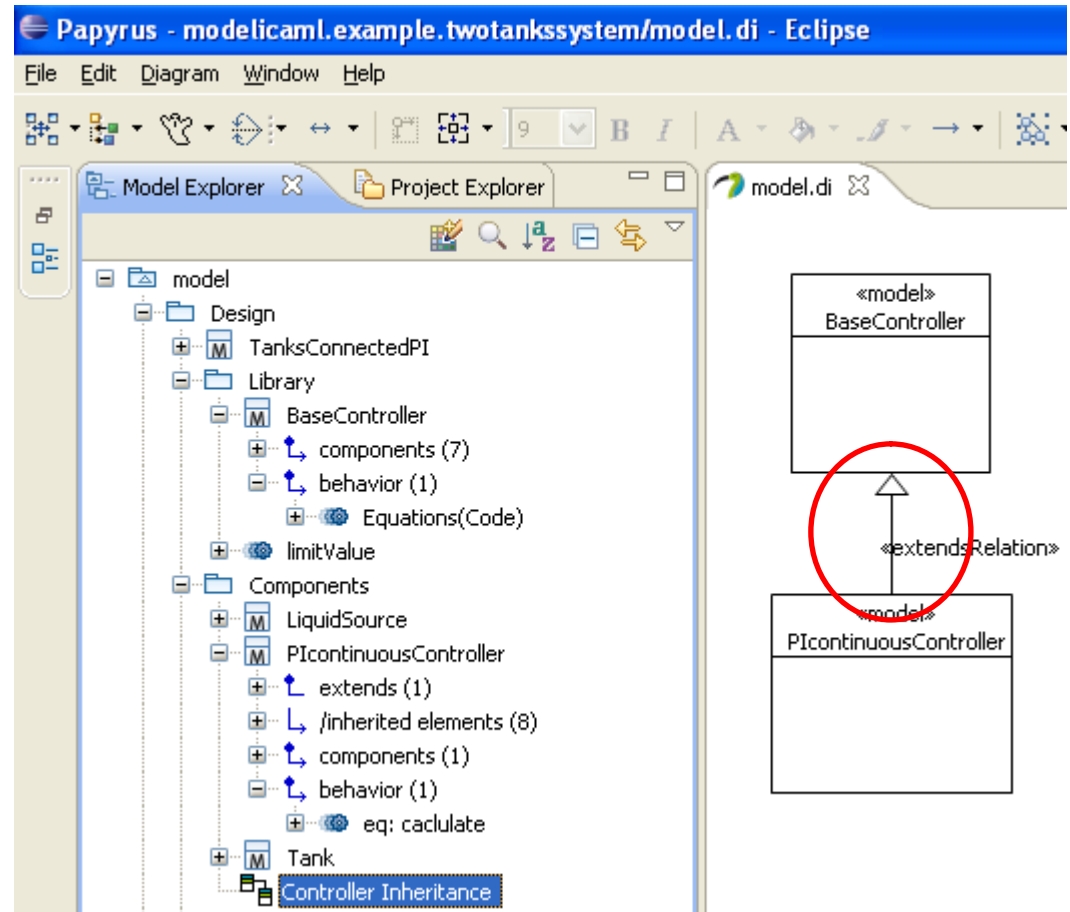
Configure Diagram Palette



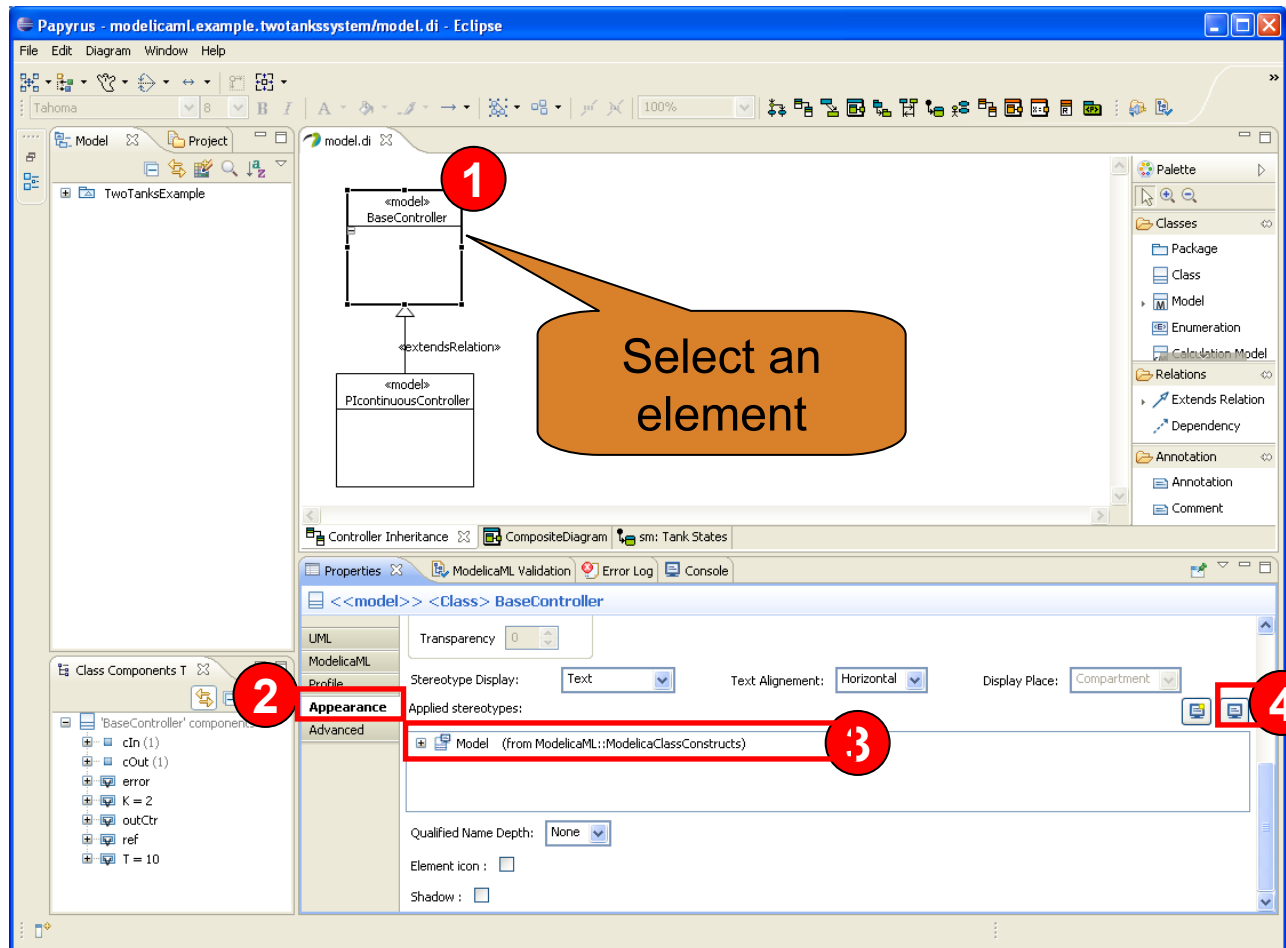
right click here...

Create Extends Relation

- Drag & drop BaseController and PIcontinuousController onto diagram
- Use the palette tool “Extends Relation”

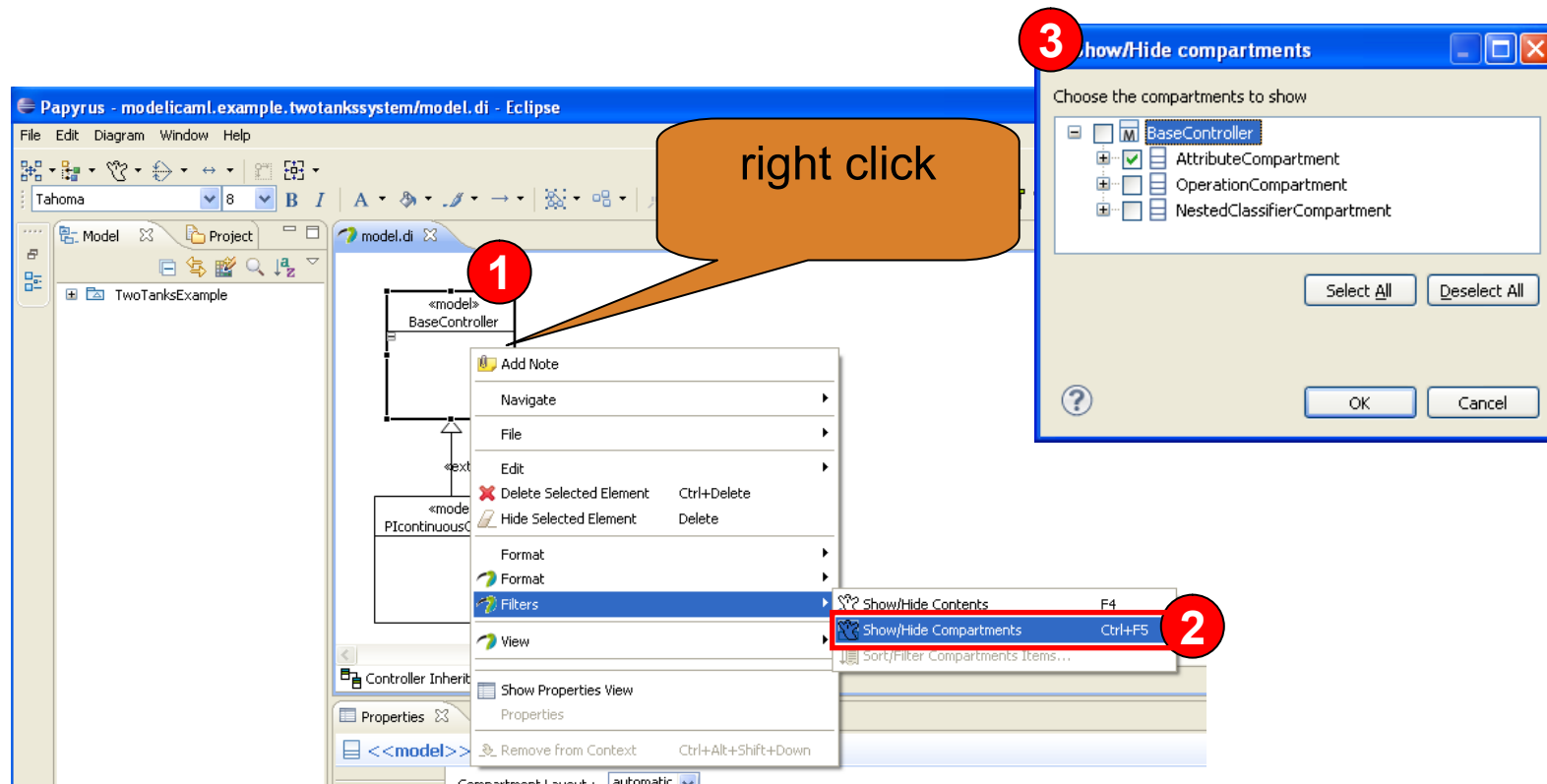


Hint: Element Appearance: Show stereotype name



The screenshot shows the Eclipse IDE interface with a UML diagram and the Properties window. The diagram displays a class hierarchy where PIcontinuousController extends BaseController. A callout box labeled '1' points to the BaseController class, and a larger callout box labeled 'Select an element' points to the same class. The Properties window is open for the selected class, showing the 'Appearance' tab. A callout box labeled '2' points to the 'Appearance' tab, and another callout box labeled '3' points to the 'Applied stereotypes' list, which contains the entry 'Model (from ModelicaML::ModelicaClassConstructs)'. A callout box labeled '4' points to the 'Applied stereotypes' list. The Properties window also shows options for 'Stereotype Display', 'Text Alignment', 'Display Place', 'Qualified Name Depth', 'Element icon', and 'Shadow'.

Hint: Element Appearance: Compartments

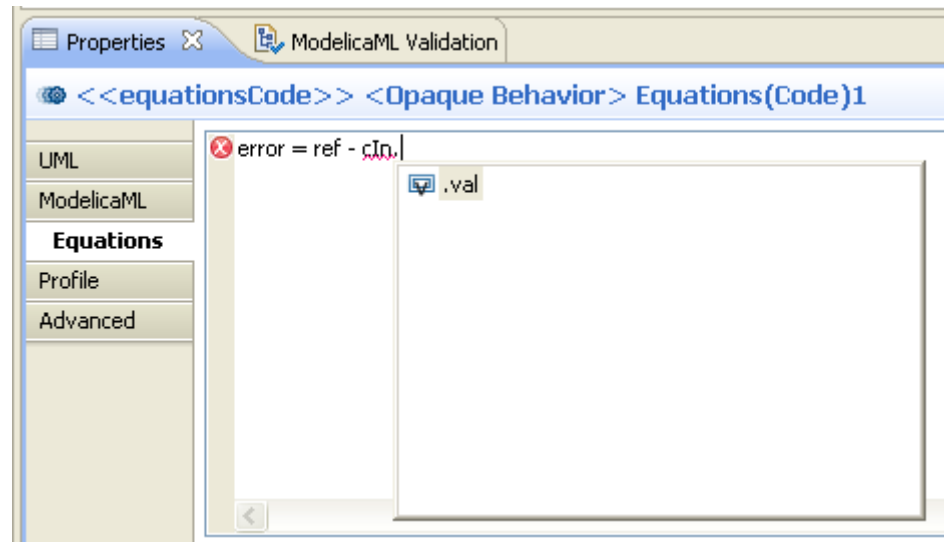


The screenshot illustrates the steps to access the 'Show/Hide Compartments' dialog in the Eclipse IDE. A right-click context menu is shown over the 'BaseController' class in the class hierarchy. The 'Filters' option is selected, which opens a sub-menu where 'Show/Hide Compartments' is highlighted. A callout bubble labeled 'right click' points to the initial right-click action. A red circle '1' marks the right-click, and a red circle '2' marks the selection of 'Show/Hide Compartments'. A third red circle '3' marks the 'Show/Hide compartments' dialog box, which is open and shows a list of compartments: 'BaseController', 'AttributeCompartment', 'OperationCompartment', and 'NestedClassifierCompartment'. The 'AttributeCompartment' is currently checked.

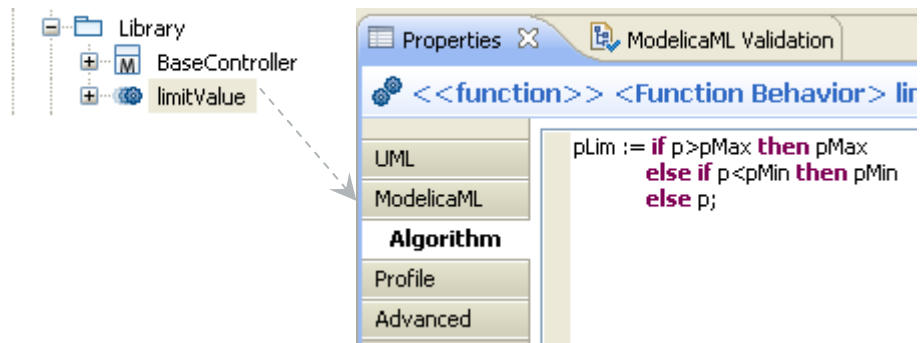
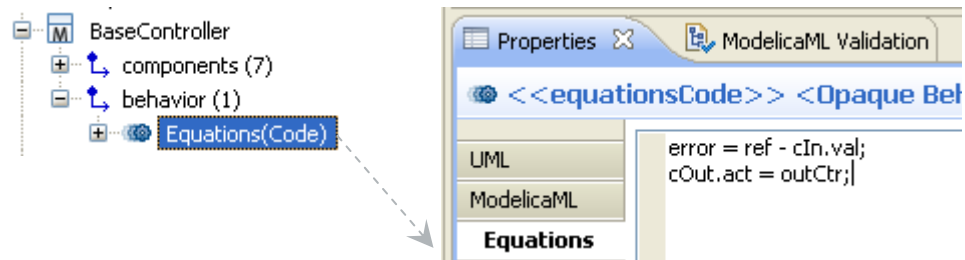
Behavior Modeling

Hint: Editing Modelica Code

- Syntax highlighting and code completion is supported in code editors
- Hit **Ctrl + Space** for code completion when editing Modelica code



Create Behavior



Create Behavior

The image displays two examples of creating behavior in a software environment. Each example consists of a component tree on the left and a corresponding property window on the right.

Example 1: LiquidSource

- Component Tree:** Shows a component named 'LiquidSource' containing two sub-components: 'components (2)' and 'behavior (1)'. The 'behavior (1)' sub-component contains an equation named 'eq: set outgoing flow level'.
- Property Window:** The 'Equations' tab is selected, showing the following code:

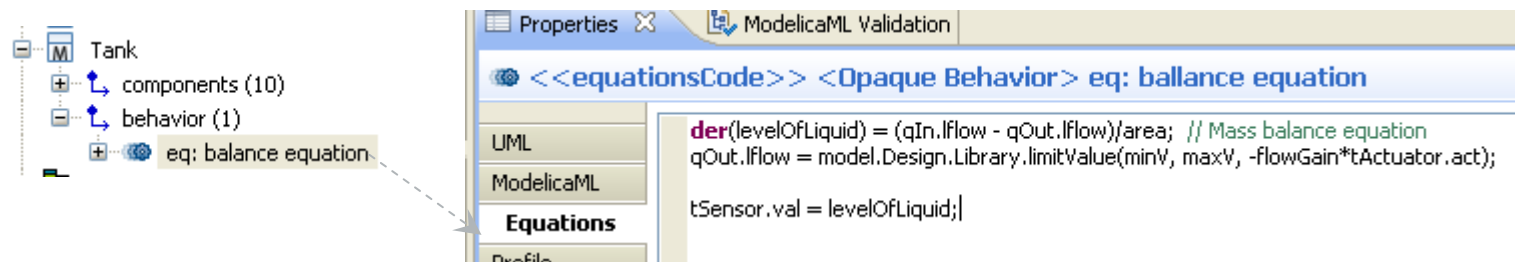

```
<<equationsCode>> <Opaque Behavior> eq: set outgoing flow level
Out.Iflow = if time > 150 then 3*flowLevel else flowLevel;
```

Example 2: PIcontinuousController

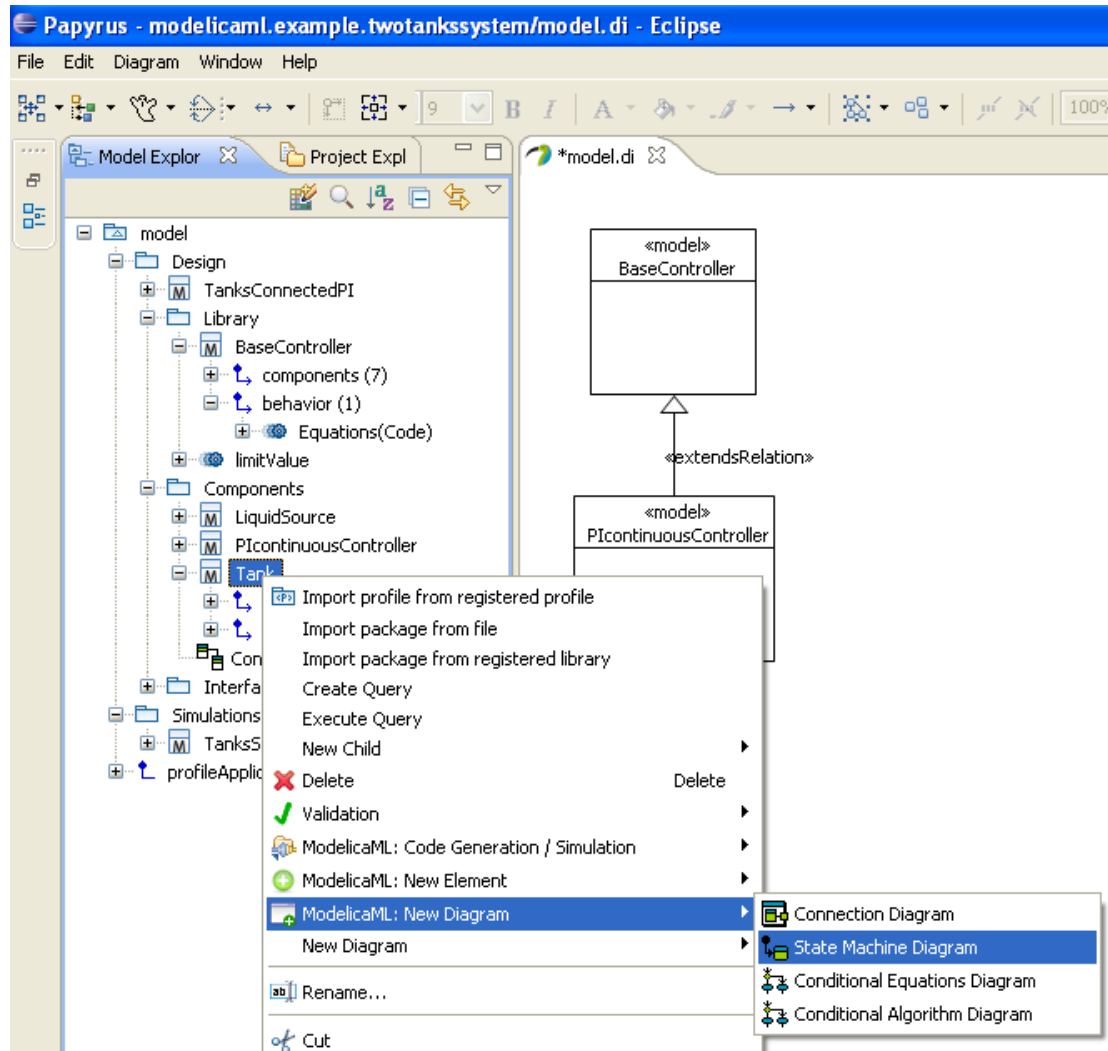
- Component Tree:** Shows a component named 'PIcontinuousController' containing two sub-components: 'components (1)' and 'behavior (1)'. The 'behavior (1)' sub-component contains an equation named 'eq: caclulate'.
- Property Window:** The 'Equations' tab is selected, showing the following code:


```
<<equationsCode>> <Opaque Behavior> eq: caclulate
der(x) = error/T;
outCtr = K*(error + x);
```

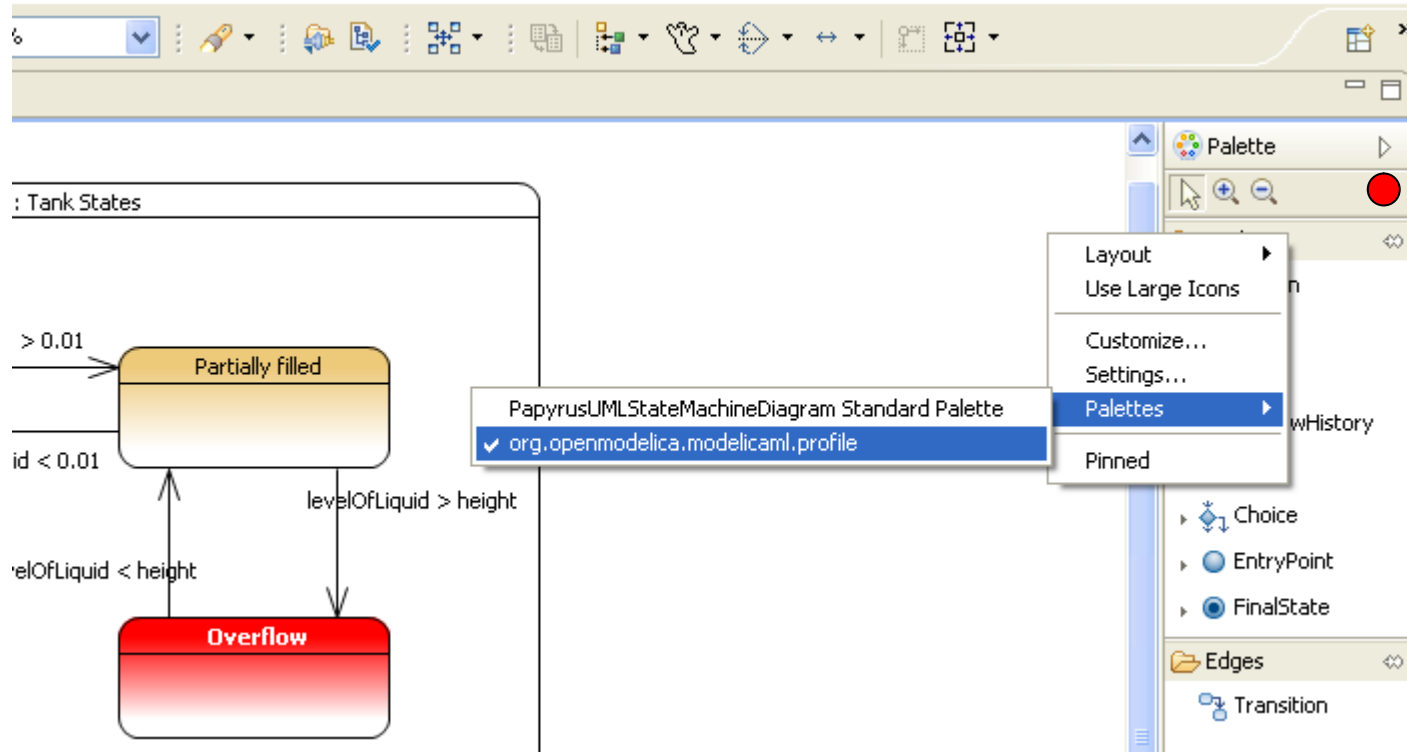
Create Behavior



Create State Machine



Configure Diagram Palette

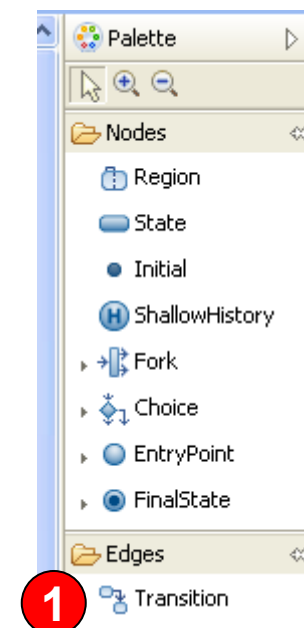
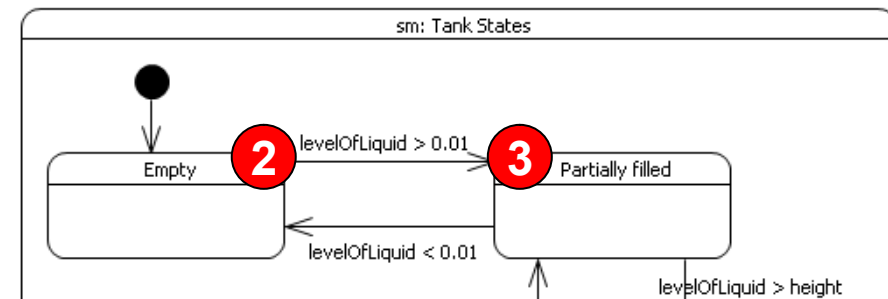


right click here...

General: Working with diagrams

State Transitions:

- 1 Select the palette tool
- 2 Click on the transition source state (click on label) and hold the mouse button
- 3 Move the mouse to the target state (to its label) and release the mouse button



Create State Machine

The screenshot shows the Papyrus IDE interface for creating a state machine. The main workspace displays a state machine diagram titled "sm: Tank States". The diagram includes three states: "Empty", "Partially filled", and "Overflow".

- Empty** (white rounded rectangle): The initial state, indicated by a black dot.
- Partially filled** (white rounded rectangle): Reached from "Empty" via the transition `levelOfLiquid > 0.01`.
- Overflow** (red rounded rectangle): Reached from "Partially filled" via the transition `levelOfLiquid > height`.
- Transitions:**
 - From "Partially filled" back to "Empty": `levelOfLiquid < 0.01`
 - From "Overflow" back to "Partially filled": `levelOfLiquid < height`

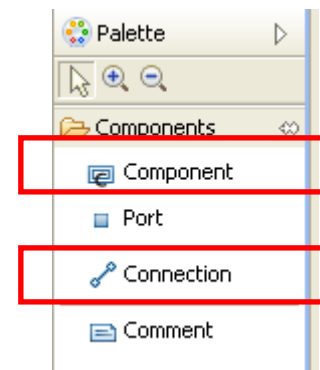
The Properties window at the bottom shows the configuration for the transition `levelOfLiquid > 0.01`, with the guard code `levelOfLiquid > 0.01`.

The left sidebar shows the Project Explorer with a tree structure including "model", "Design", "TanksConnectedPI", "Library", "BaseController", "Components", "Tank", and "sm: Tank States". The bottom-left pane shows the Class Components Tree for the "Tank" component, listing attributes like `area`, `flowGain`, `height`, `levelOfLiquid`, and `maxV`.

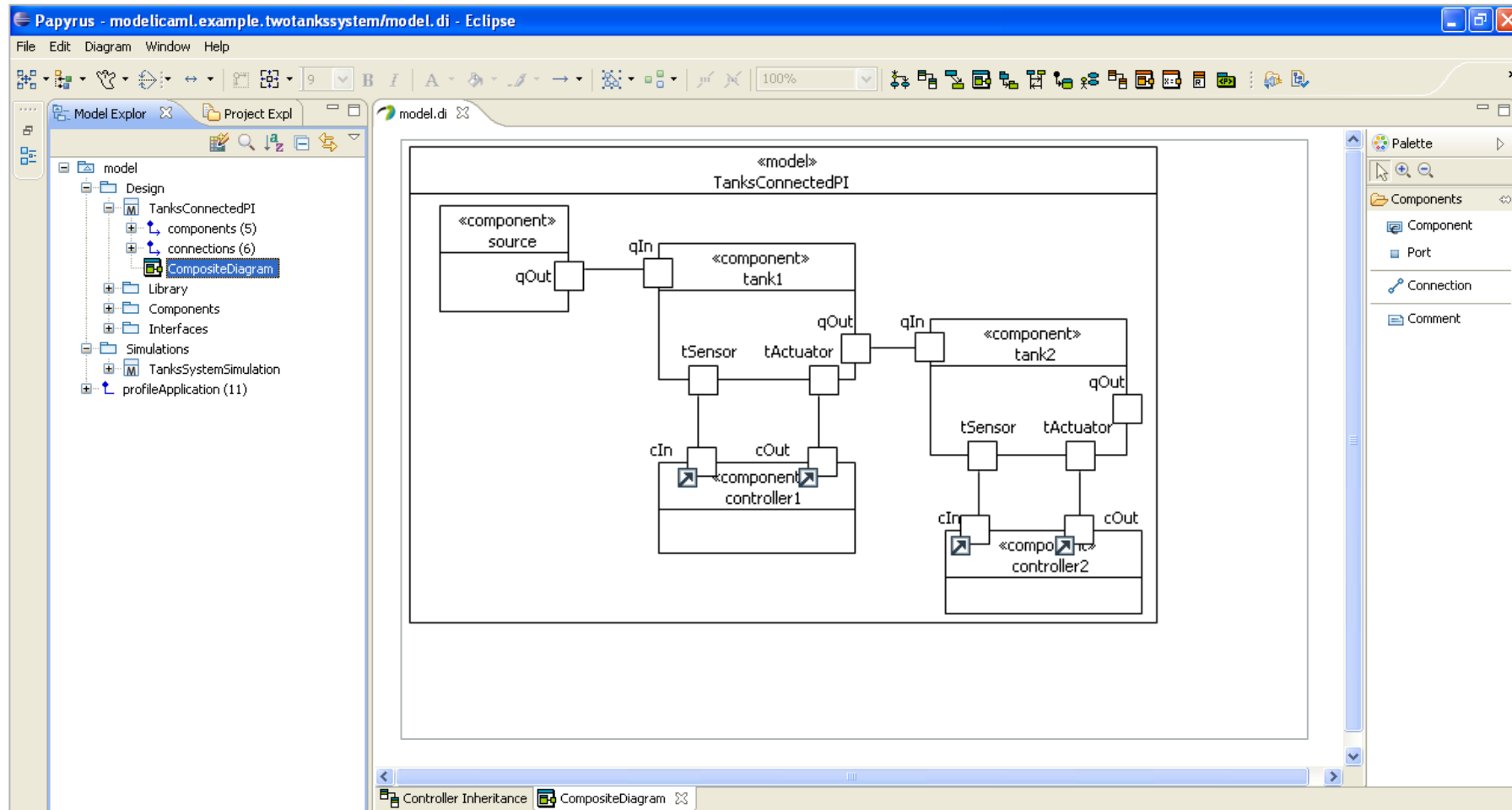
Architecture Modeling

Create Connection Diagram

- Create a ModelicaML Connection Diagram under the TanksConnectedPI class
- Use components tool from the palette to create components inside the class on the diagram
- Define the types of components
- Use Model Explorer to find the ports
- Drag&Prop ports into respective components
- Arrange the components
- Use the “Connection” tool from the palette for connecting ports



Model System Architecture



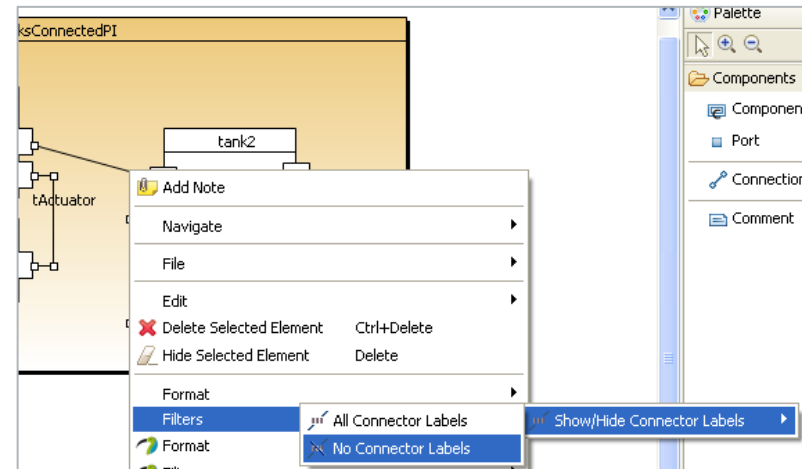
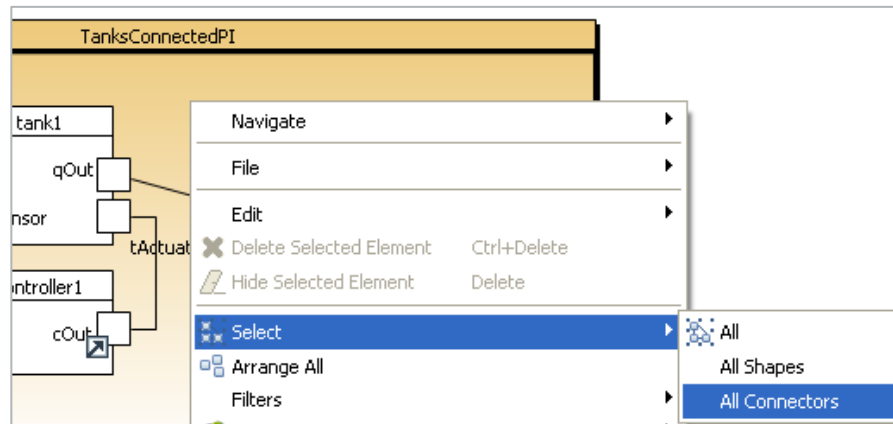
Create Connection Diagram

The screenshot shows the Papyrus IDE interface with the following elements and annotations:

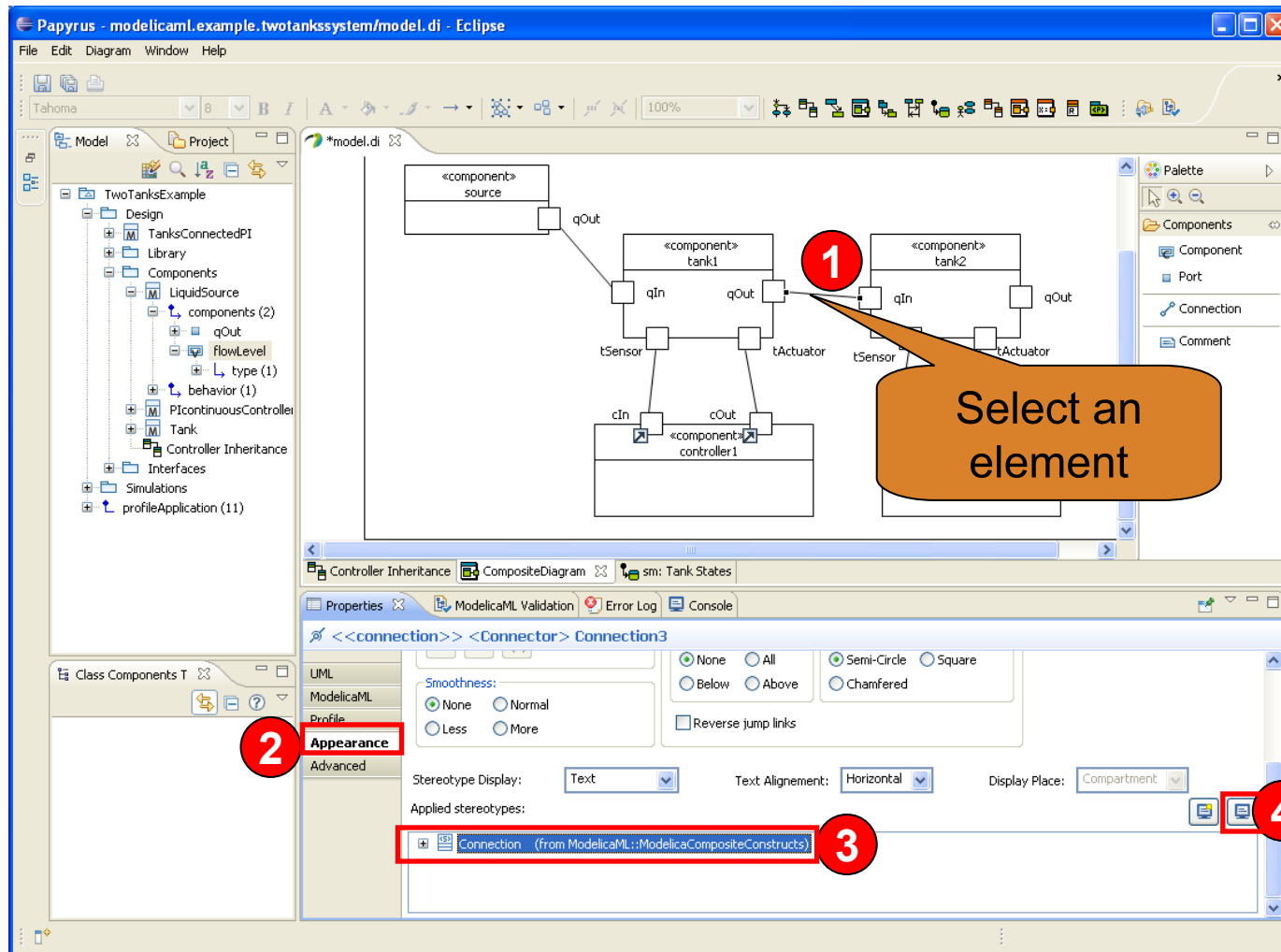
- Model Explorer:** Shows the project structure for 'TwoTanksExample'. The 'qOut' port under 'LiquidSource' is highlighted with a red box and labeled with **3** 'Drag&drop the port into component'.
- Diagram:** A UML Component Diagram for 'TanksConnectedPI' is shown. A new component box labeled 'source' is being created, with a red box around it and label **1** 'Create components using the palette tools'. The diagram also shows existing components like 'tank1', 'tank2', and 'controller1' with their ports and connections.
- Properties View:** The 'source' component's properties are shown. The 'Type' field is set to '<<model>> <<Class>> LiquidSource' and is highlighted with a red box and label **2** 'Set component type'.
- Palette:** The 'Components' palette is visible on the right, with a red dashed line pointing to the 'source' component in the diagram, labeled with **4** 'Connect components'.

Hint: Hide the name of all connectors

- Click on the compartment of the class
- Right-click -> “Select” -> “All Connectors”
- Right-click on one of the selected connectors -> “Filters” -> “Show/Hide connector Labels” -> “No connector Labels”



Hint: Element Appearance: Hide the name of the connection stereotype



The screenshot shows the Eclipse IDE with a UML diagram of a two-tank system. A callout bubble with the text "Select an element" points to a connection line between two tanks, with a red circle containing the number "1" next to it. The Properties window is open, showing the "Appearance" tab for the selected connection. A red box highlights the "Appearance" tab, with a red circle containing the number "2" next to it. Another red box highlights the "Connection" stereotype in the "Applied stereotypes" list, with a red circle containing the number "3" next to it. A fourth red circle containing the number "4" is located at the bottom right of the Properties window.

Component Modifications

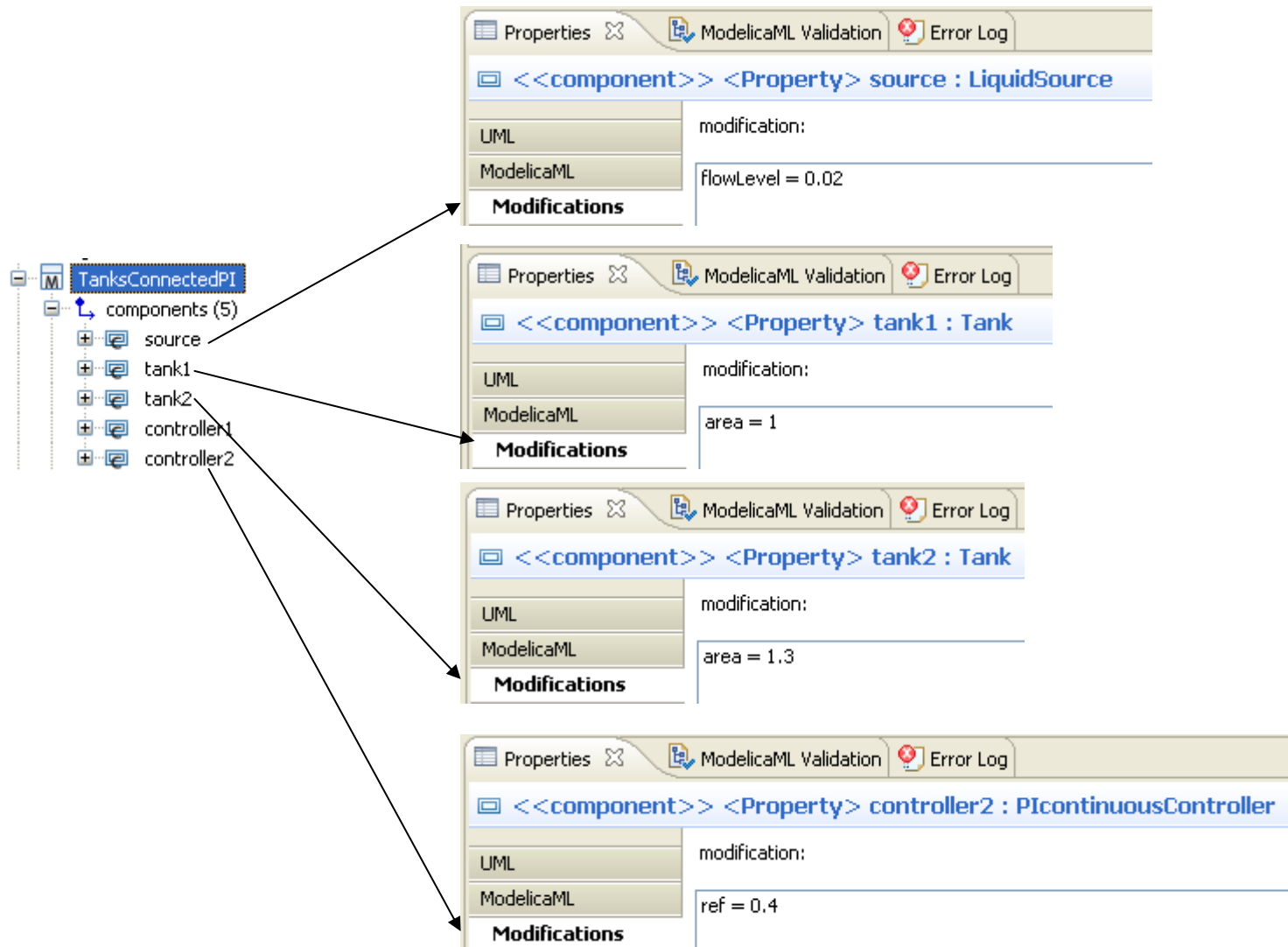


Define Component Modifications

The screenshot illustrates the process of defining component modifications in ModelicaML. It features three main windows:

- Model Explorer:** Shows a project tree for 'TwoTanksExample'. Under 'Design' > 'TanksConnectedPI', the 'components (5)' folder is expanded. 'controller1' is selected and highlighted with a red circle labeled '1'.
- modification editor:** A dialog box with a text field containing 'ref = 0.25' and 'OK'/'Cancel' buttons. A red circle labeled '3' is positioned near the text field, and another red circle labeled '4' is near the 'OK' button.
- Properties Window:** Shows the 'Modifications' tab for 'controller1 : PIcontinuousController'. The 'ModelicaML' section contains the text 'ref = 0.25'. A red circle labeled '2' is positioned near the 'Edit', '+', and '-' icons to the right of the text field.

Define Component Modifications



Model Validation



Validate Model

The screenshot shows the Eclipse IDE with the ModelicaML Modeling environment. The main window displays a block diagram of a two-tank system named 'TanksConnectedPI'. The diagram includes a 'source' block with a 'qOut' port connected to the 'qIn' port of 'tank1'. 'tank1' has a 'tSensor' and a 'tActuator' port connected to 'tank2'. 'tank2' has its own 'tSensor' and 'tActuator' ports. Both tanks have 'cIn' and 'cOut' ports. The diagram is connected to 'controller1' and 'controller2' blocks.

The left sidebar shows the 'Model Explorer' with a tree view of the project structure. The 'TwoTanksExample' folder is expanded, showing sub-folders for 'Requirements', '_ValueBindings', 'Design', and 'Library'. The 'Design' folder contains 'TanksConnectedPI' and 'TanksConnectedPID'. The 'Library' folder contains 'LiquidSource', 'PIcontinuousController', 'PIDcontinuousController', and 'Tank'. The 'Components' folder contains various components like 'ReadSignal tSensor', 'ActSignal tActuator', 'LiquidFlow qIn', 'LiquidFlow qOut', and several parameters and equations.

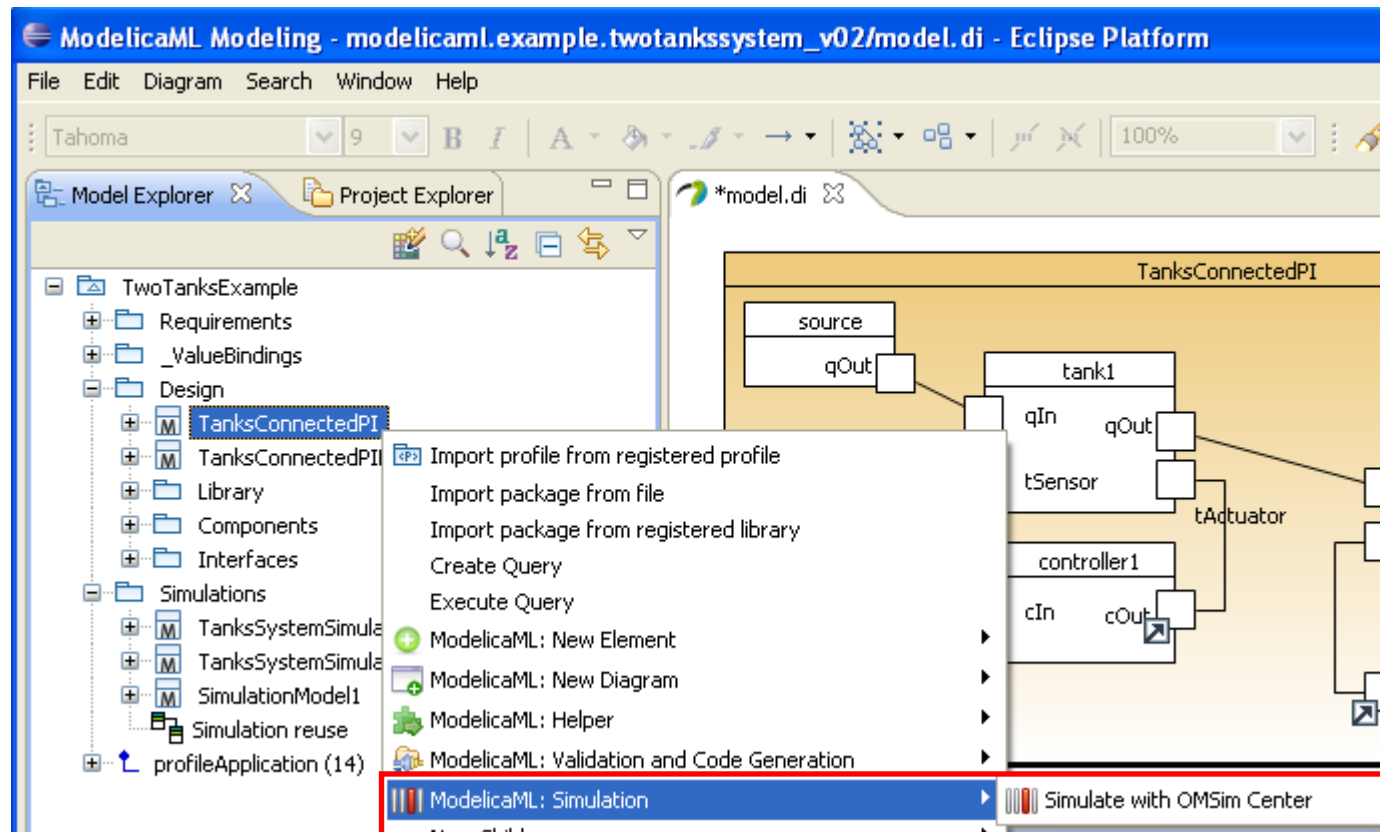
The bottom panel shows the 'ModelicaML Validation' window, which is highlighted with a red border. It displays a table with one item indicating that no errors were detected for the 'TwoTanksExample' model.

Description	Location
ModelicaML validation: No errors were detected for 'TwoTanksExample'.	TwoTanksExample

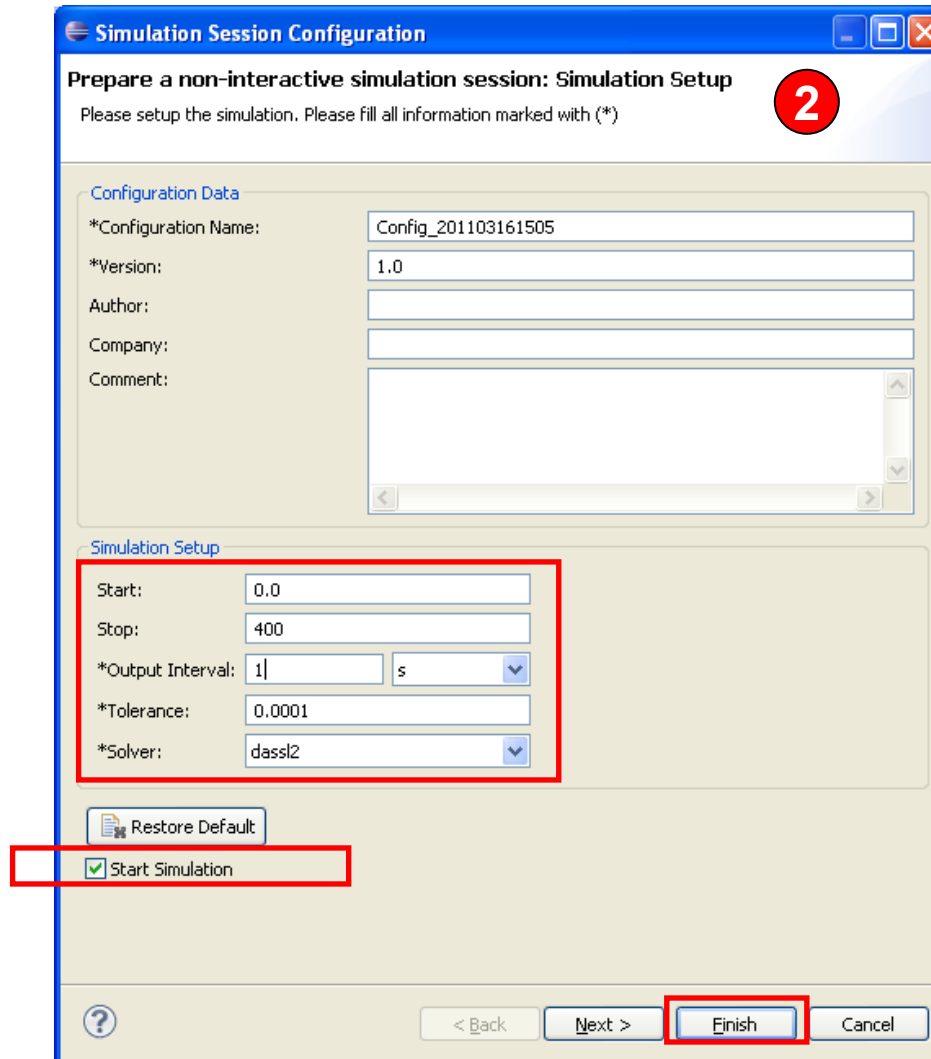
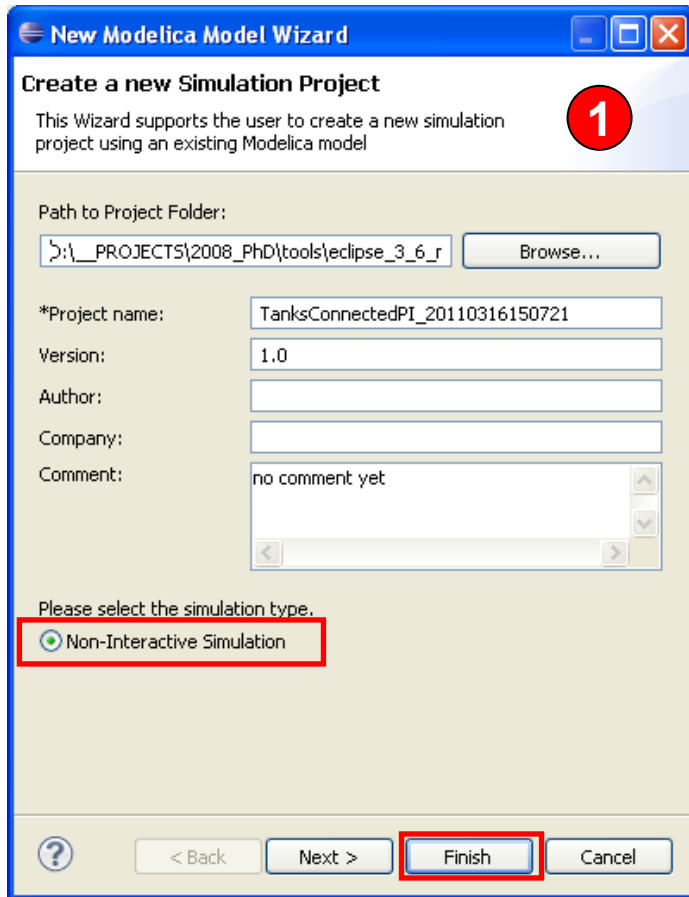
Model Simulation



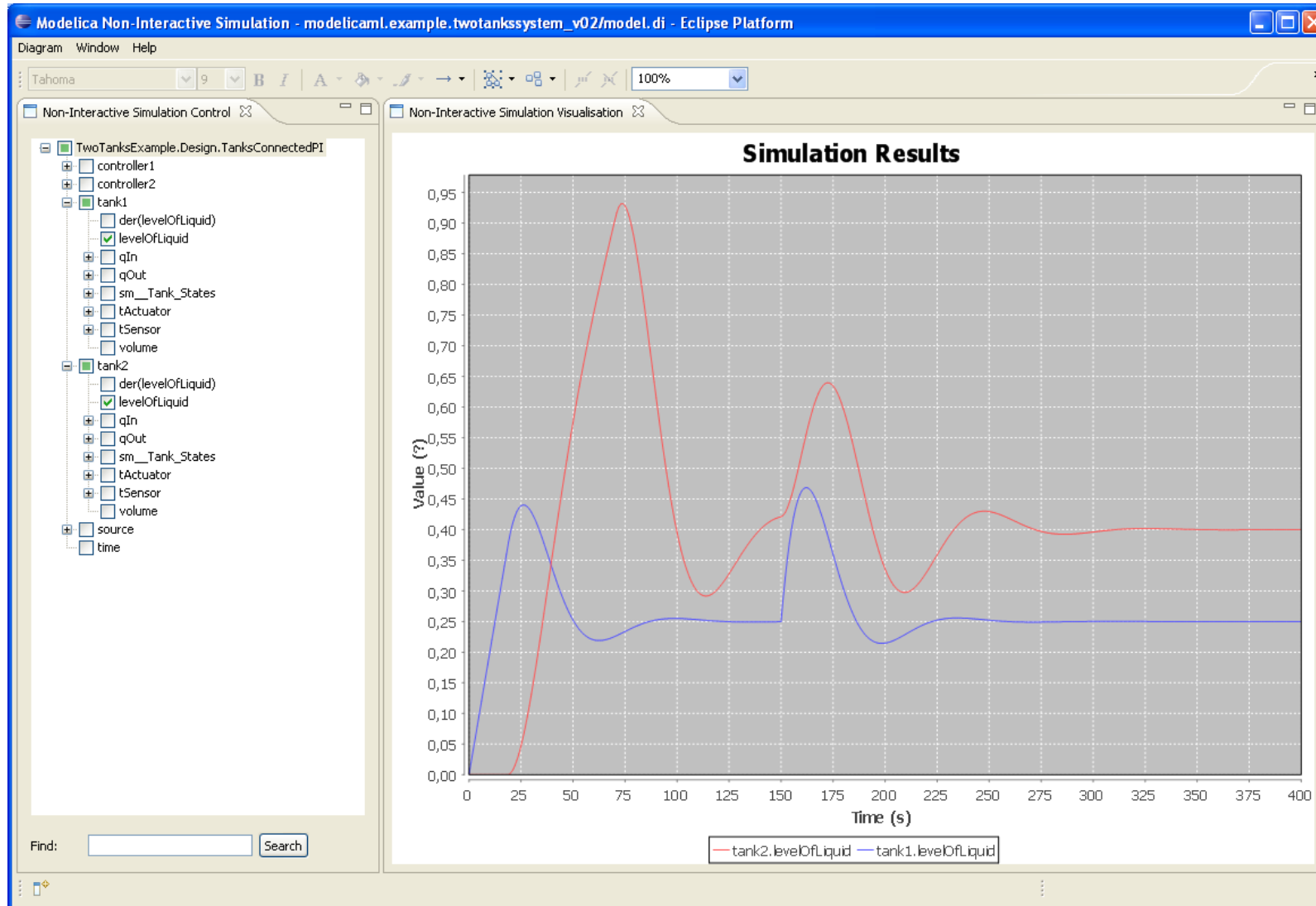
Simulate Model



Simulate Model

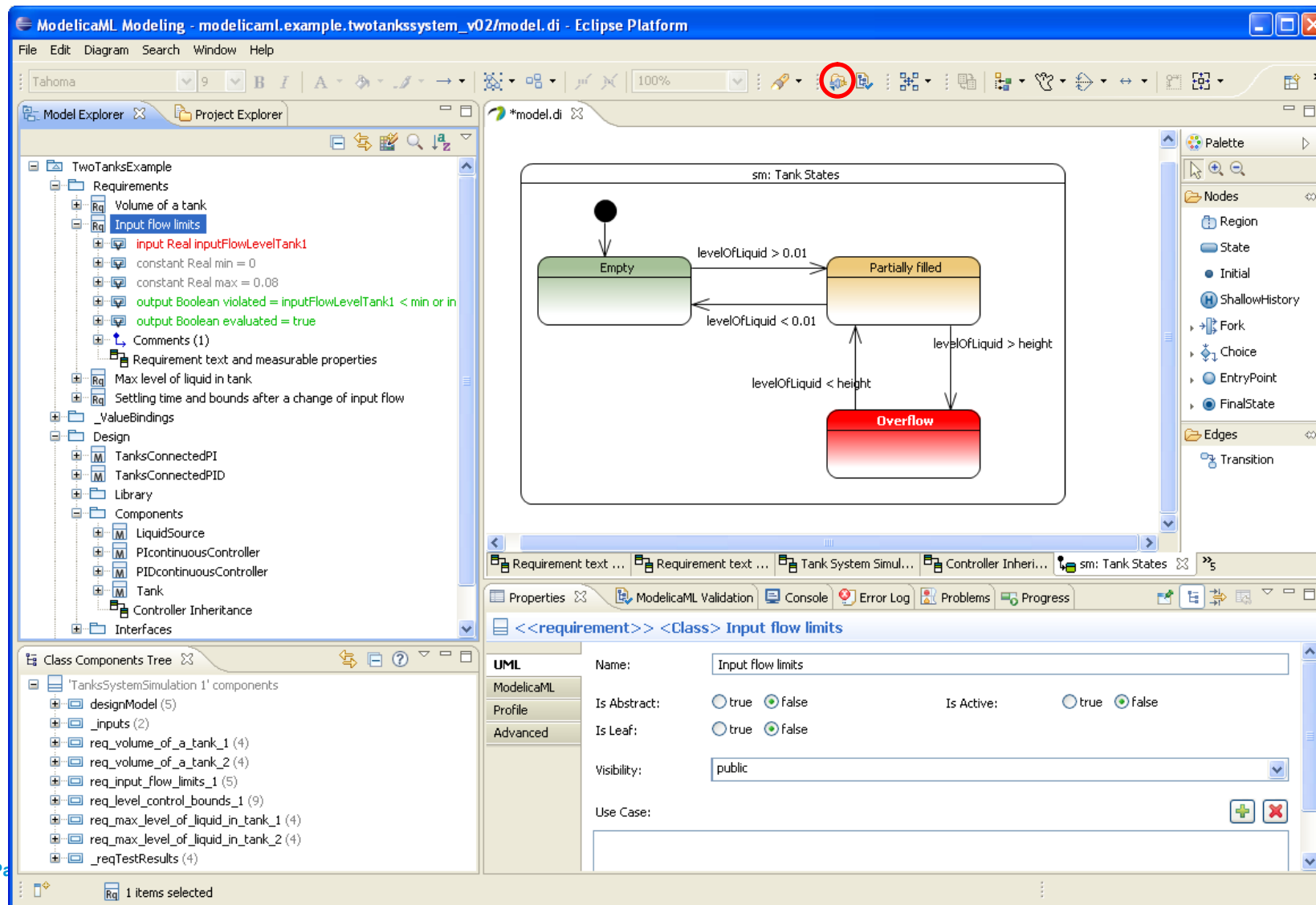


Simulate Model



Modelica Code Generation

Launch Modelica Code Generation



The screenshot shows the ModelicaML Eclipse IDE interface. The main window displays a state machine diagram titled "sm: Tank States" with three states: "Empty", "Partially filled", and "Overflow". Transitions are labeled with conditions on "levelOfLiquid".

```

stateDiagram-v2
    [*] --> Empty
    Empty --> PartiallyFilled : levelOfLiquid > 0.01
    PartiallyFilled --> Empty : levelOfLiquid < 0.01
    PartiallyFilled --> Overflow : levelOfLiquid > height
    Overflow --> PartiallyFilled : levelOfLiquid < height
  
```

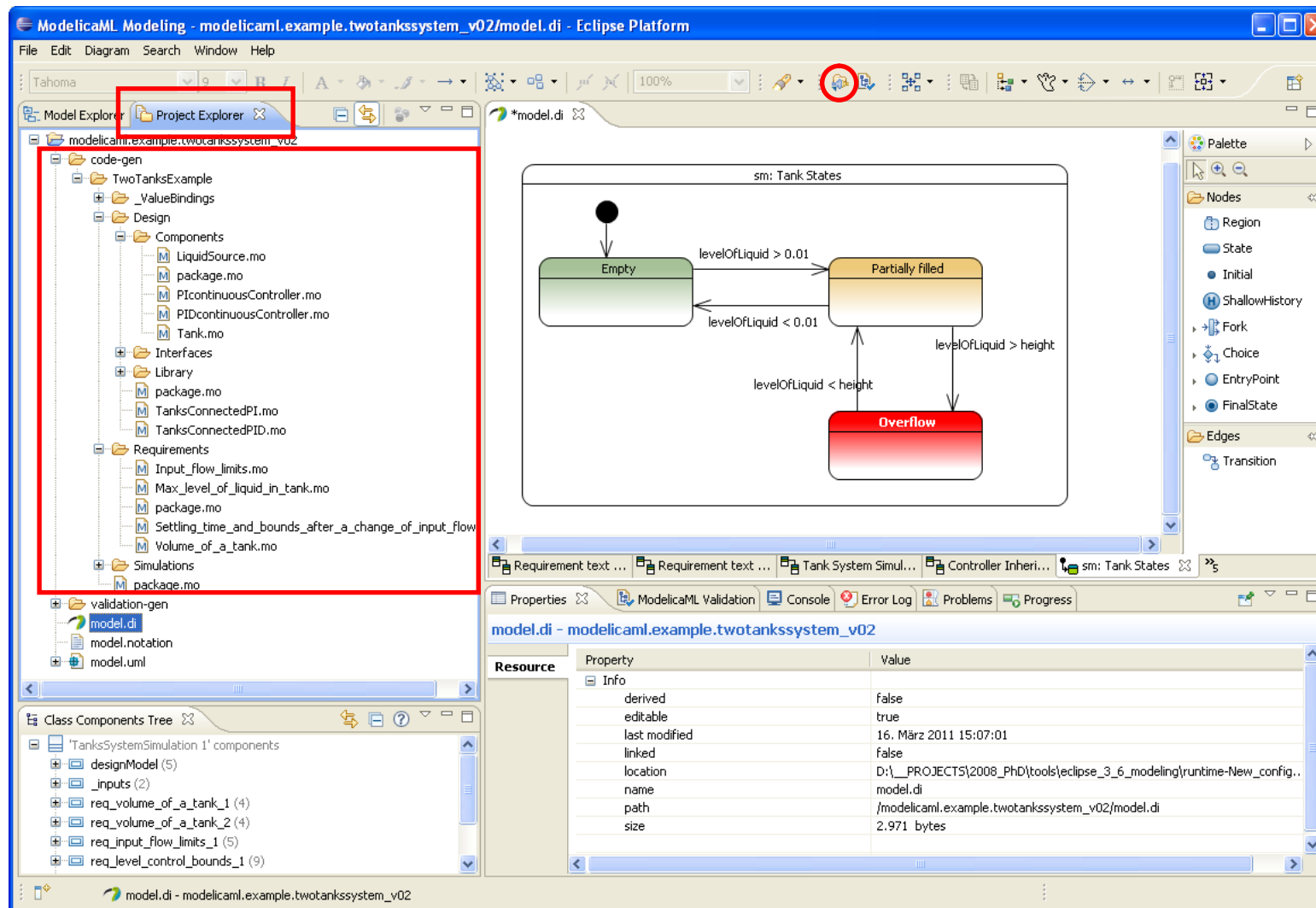
The left sidebar shows the "Model Explorer" with a tree view of the project "TwoTanksExample". The "Class Components Tree" at the bottom left lists various components like "designModel", "_inputs", and "req_volume_of_a_tank_1".

The bottom right pane shows the "Properties" view for the class "Input flow limits".

UML	Name:	Input flow limits
ModelicaML	Is Abstract:	<input type="radio"/> true <input checked="" type="radio"/> false
Profile	Is Active:	<input type="radio"/> true <input checked="" type="radio"/> false
Advanced	Is Leaf:	<input type="radio"/> true <input checked="" type="radio"/> false
	Visibility:	public
	Use Case:	

The toolbar at the top contains various icons, with the "Generate" icon (a document with a lightning bolt) circled in red.

Generated Modelica Code



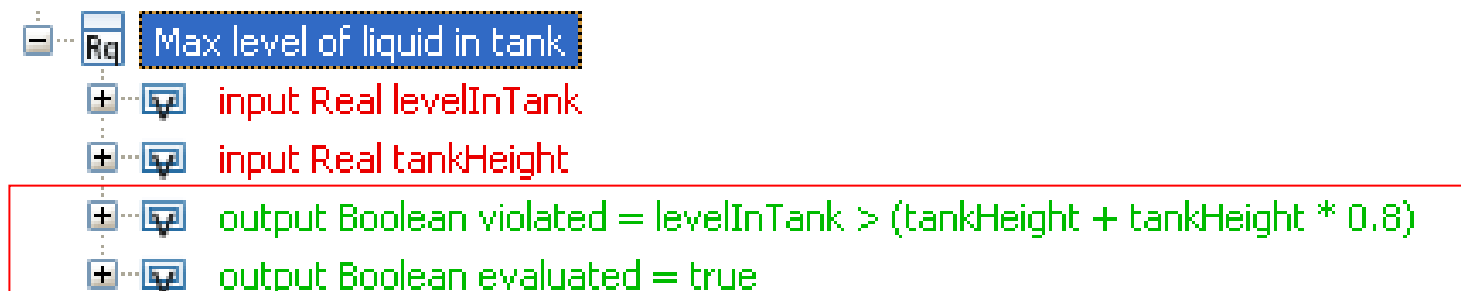
The screenshot shows the Eclipse IDE interface for ModelicaML Modeling. The Project Explorer on the left is highlighted with a red box, showing the 'code-gen' directory containing generated Modelica code files. The main editor displays a state machine diagram for 'Tank States' with states 'Empty', 'Partially filled', and 'Overflow'. The Properties window at the bottom right shows details for the 'model.di' resource.

Resource	Property	Value
model.di	Info	
	derived	false
	editable	true
	last modified	16. März 2011 15:07:01
	linked	false
	location	D:\..._PROJECTS\2008_PhD\tools\eclipse_3_6_modeling\runtime-New_config..
	name	model.di
	path	/modelicaml.example.twotankssystem_v02/model.di
size	2.971 bytes	

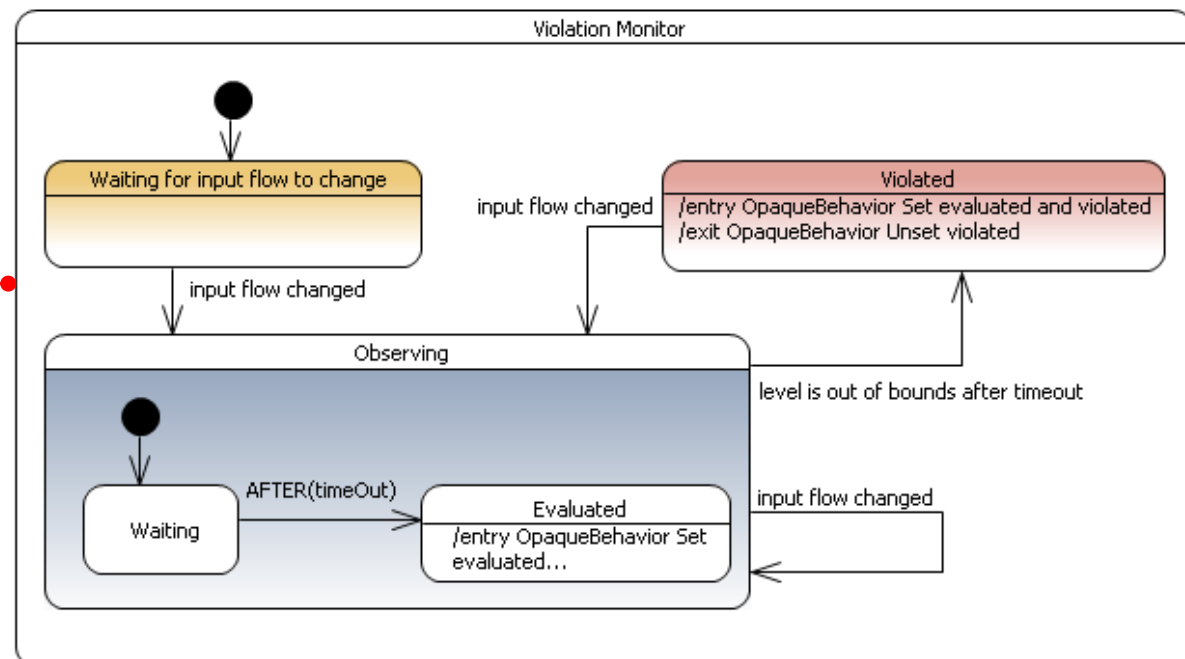
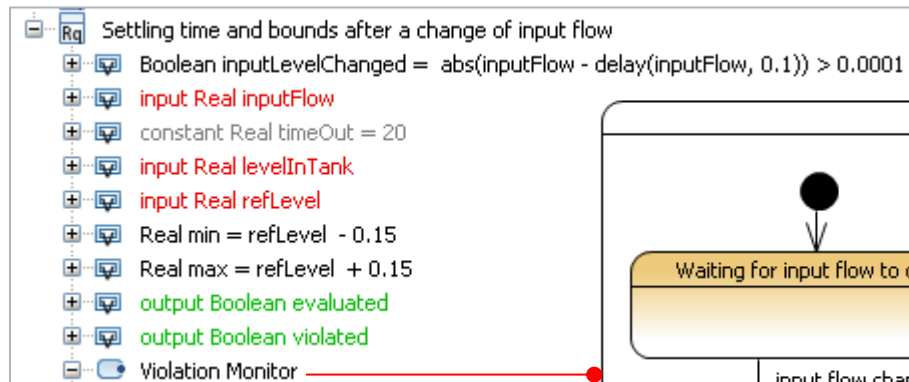
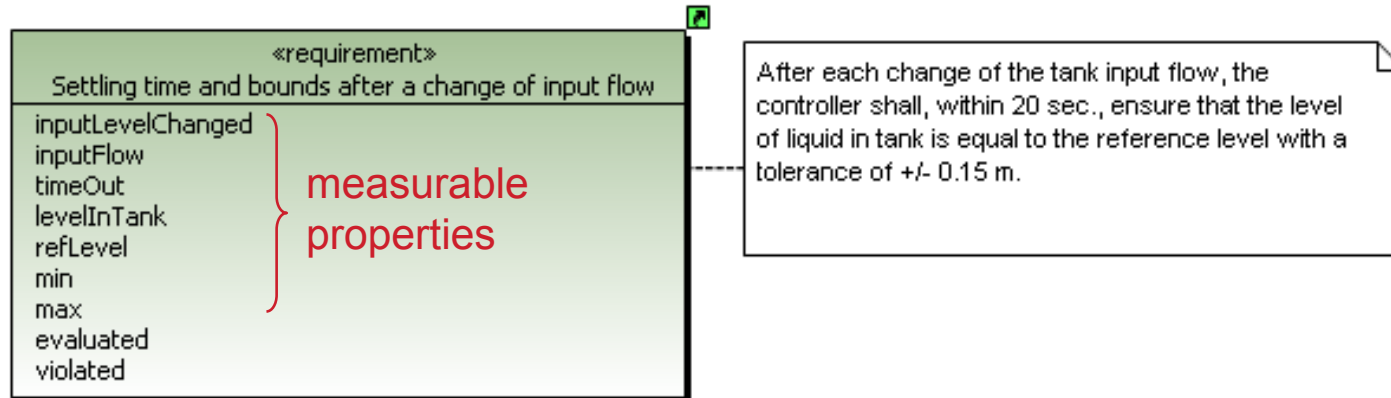
Handling of System Requirements

Formalize a Textual Requirements (Example 1)

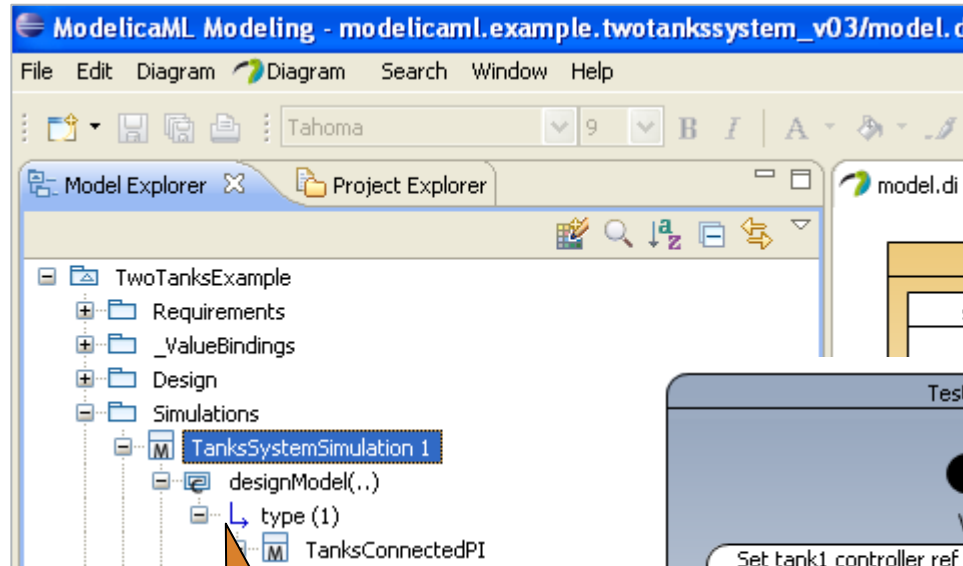
1. Identify and formalize measurable properties addressed in the requirement statement
2. Define when requirement is evaluated (pre-condition) and violated



Formalize a Textual Requirements (Example 2)

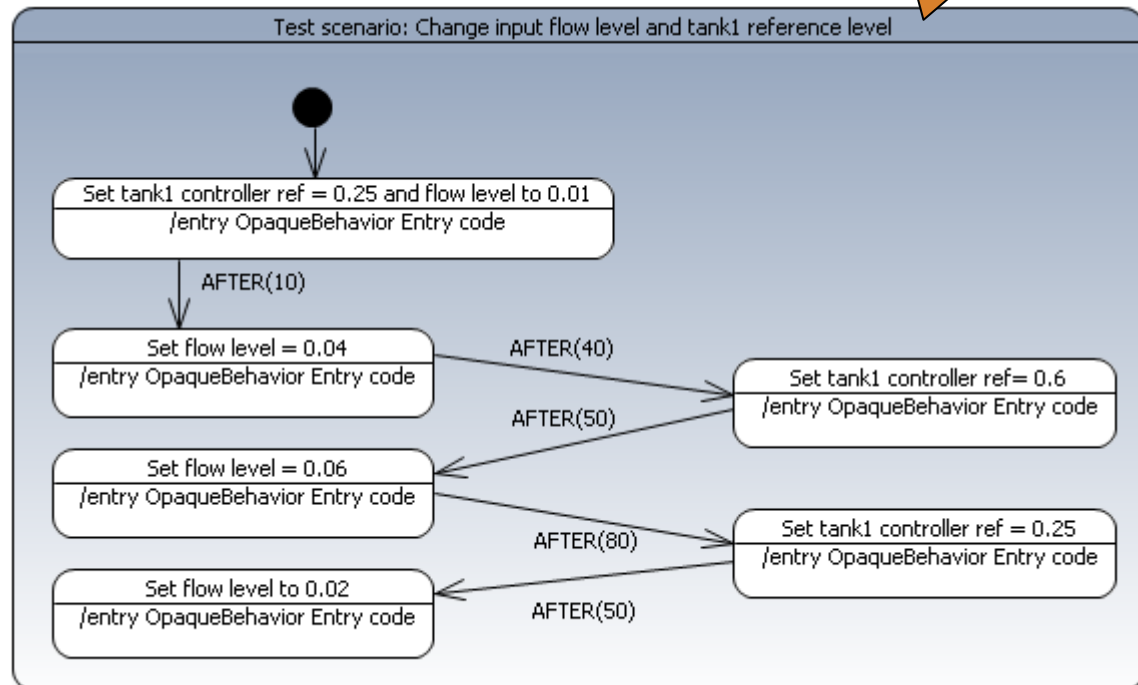


Instantiate Design Model

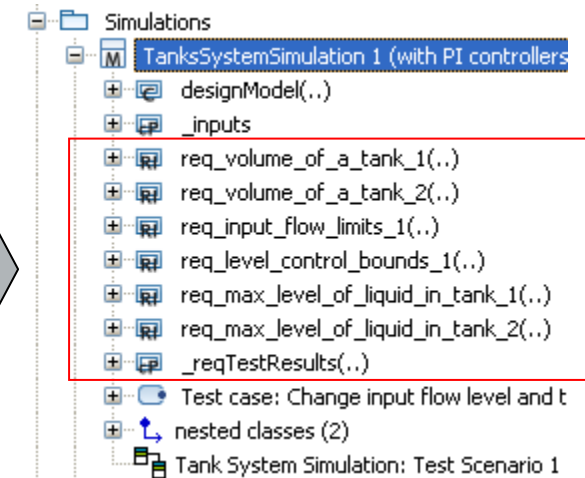
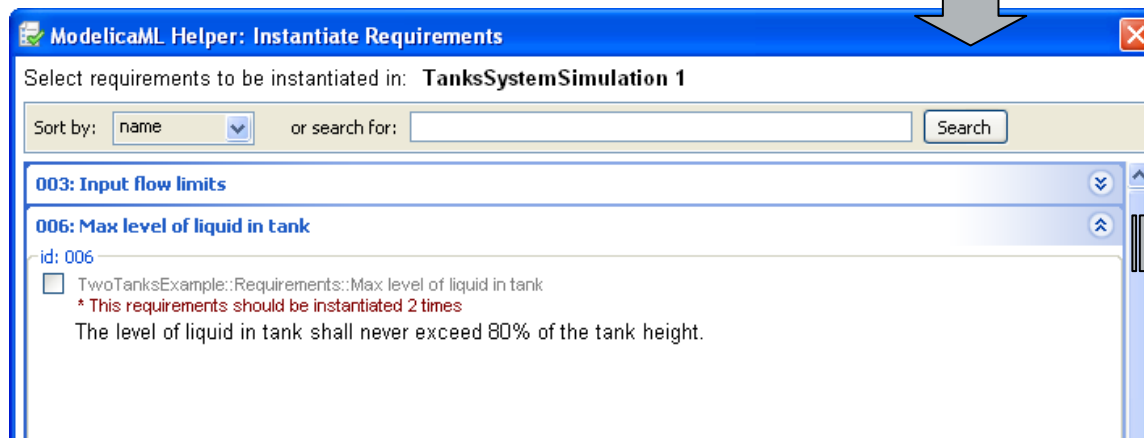
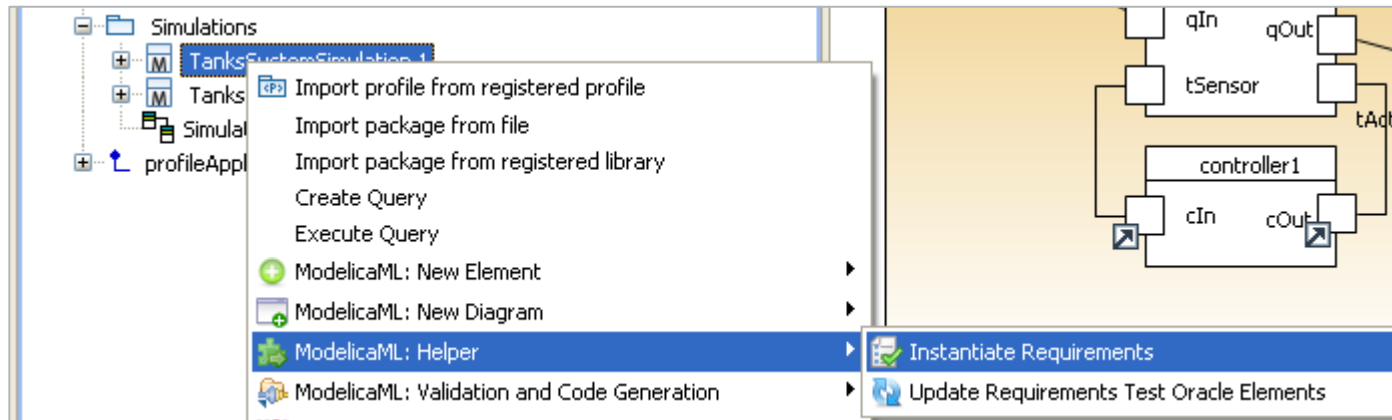


instantiated design model

Test scenario that stimulates the design model



Instantiate Requirements



Bind Design Inputs and Requirement Properties

Stimuli and parameters for design model:

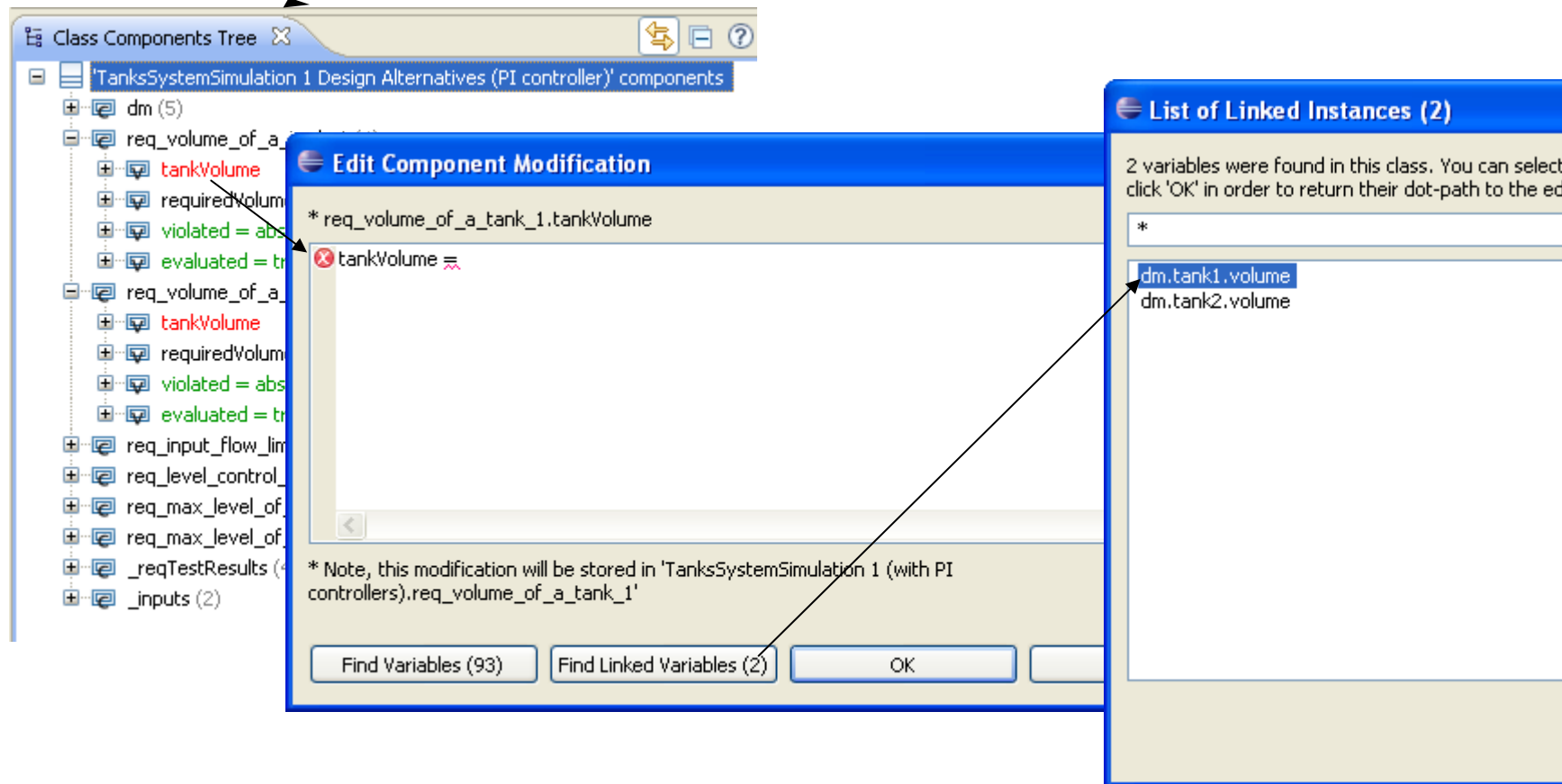
```
controller1.ref = _inputs.dm_controller1_ref
source.flowLevel = _inputs.dm_source_flowLevel
```

Example for requirement properties binding:

```
inputFlow = designModel.tank1.qIn.lflow
levelInTank = designModel.tank1.levelOfLiquid
refLevel = designModel.controller1.ref
```

Finding Properties that are Linked Through Mediators

Instantiates a class and enables the finding of different properties that are linked through mediators




Using Mediators for Value Binding

Bind Design Inputs and Requirement Properties

Challenge:


- When requirements and design are instantiated in a test model for simulations their properties need to be bound (i.e. requirement properties must access the right values from the design model)
- Is it possible to support or even automate the binding?

Questions:

-  - How to find the right design model properties, that requirements properties shall be bound to, without having a deep knowledge of the design model?

```

req_level_control_bounds_1 (3)
├── inputFlow = designModel.tank1.qIn.Iflow
├── levelInTank = designModel.tank1.levelOfLi ...
└── refLevel = designModel.controller1.ref
    
```

-  - A requirement may be needed to be instantiated multiple times (e.g. for each tank or controller). How to know how many times a requirement should be instantiated and what are the respective bindings to the design model properties?

```

req_max_level_of_liquid_in_tank_1 (2)
├── levelInTank = designModel.tank1.levelOfLi ...
├── tankHeight = designModel.tank1.height
└── req_max_level_of_liquid_in_tank_2 (2)
    ├── levelInTank = designModel.tank2.levelOfLi ...
    └── tankHeight = designModel.tank2.height
    
```

Bind Design Inputs and Requirement Properties

Mediators Concept:

- ValueMediator is a ModelicaML concept that allows to define a n:n relation between properties of different classes.
 - This is done at the class property level and enables the finding of linked properties when they are instantiated in the same class hierarchy (e.g. in the same simulation model)
 - ValueMediator are only used at modeling level and do not affect the code generation
- ValueProvider provides the value
- ValueClient obtains the value

- When formalizing requirements the requirements analyst creates and reuses ValueMediators and defines which requirement property should obtain the value from which ValueMediator
- When creating the design model (based on requirements) the designer defines which design model properties provide value for which ValueMediator

Using Mediators

Requirement

- Rq **Setting time and bounds after a change of input flow**
- Boolean inputLevelChanged = $\text{abs}(\text{inputFlow} - \text{delay}(\text{inputFlow}, 0.1))$
- input Real inputFlow
- constant Real timeOut = 20
- input Real levelInTank
- input Real refLevel
- Real min = $\text{refLevel} - 0.15$
- Real max = $\text{refLevel} + 0.15$
- output Boolean evaluated
- output Boolean violated

The screenshot shows the 'ModelicaML Validation' window. The main pane displays the configuration for the property `levelInTank : ModelicaReal`. The 'Applied stereotypes' list includes:

- Variable (from ModelicaML::ModelicaCompositeConstructs)
- ValueClient (from ModelicaMLTesting::ValueBinding)
- obtainsValueFrom: ValueMediator [0..*] = [levelOfLiquidInTankMediator]**

 The 'Profile' tab is selected and highlighted with a red box. Arrows indicate the mapping from the requirement's `levelInTank` input to the `ValueClient` stereotype and the `obtainsValueFrom` configuration.

Design Artifact

- M Tank
 - ReadSignal tSensor
 - ActSignal tActuator
 - LiquidFlow qIn
 - LiquidFlow qOut
 - parameter Real flowGain = 0.05
 - parameter Real minV = 0
 - parameter Real maxV = 10
 - parameter Real area = 1
 - parameter Real height = 2
 - Real volume = $\text{area} * \text{height}$
 - Real levelOfLiquid
 - eq: balance equation
 - sm: Tank States

Mediator

levelOfLiquidInTankMediator

The screenshot shows the configuration for the `levelOfLiquidInTankMediator` value provider. The 'Applied stereotypes' list includes:

- ValueProvider (from ModelicaMLTesting::ValueBinding)
- providesValueFor: ValueMediator [0..*] = [levelOfLiquidInTankMediator]**

 An arrow points from the 'Tank' design artifact's `levelOfLiquid` property to this configuration. Another arrow points from the `ValueMediator` in this configuration to the `ValueMediator` in the requirement configuration above.

Why Using Mediators?

Support for the access of design properties values:

- ValueMediator is an interfaces the design model provides in order to enable support for requirements properties to the right design properties.

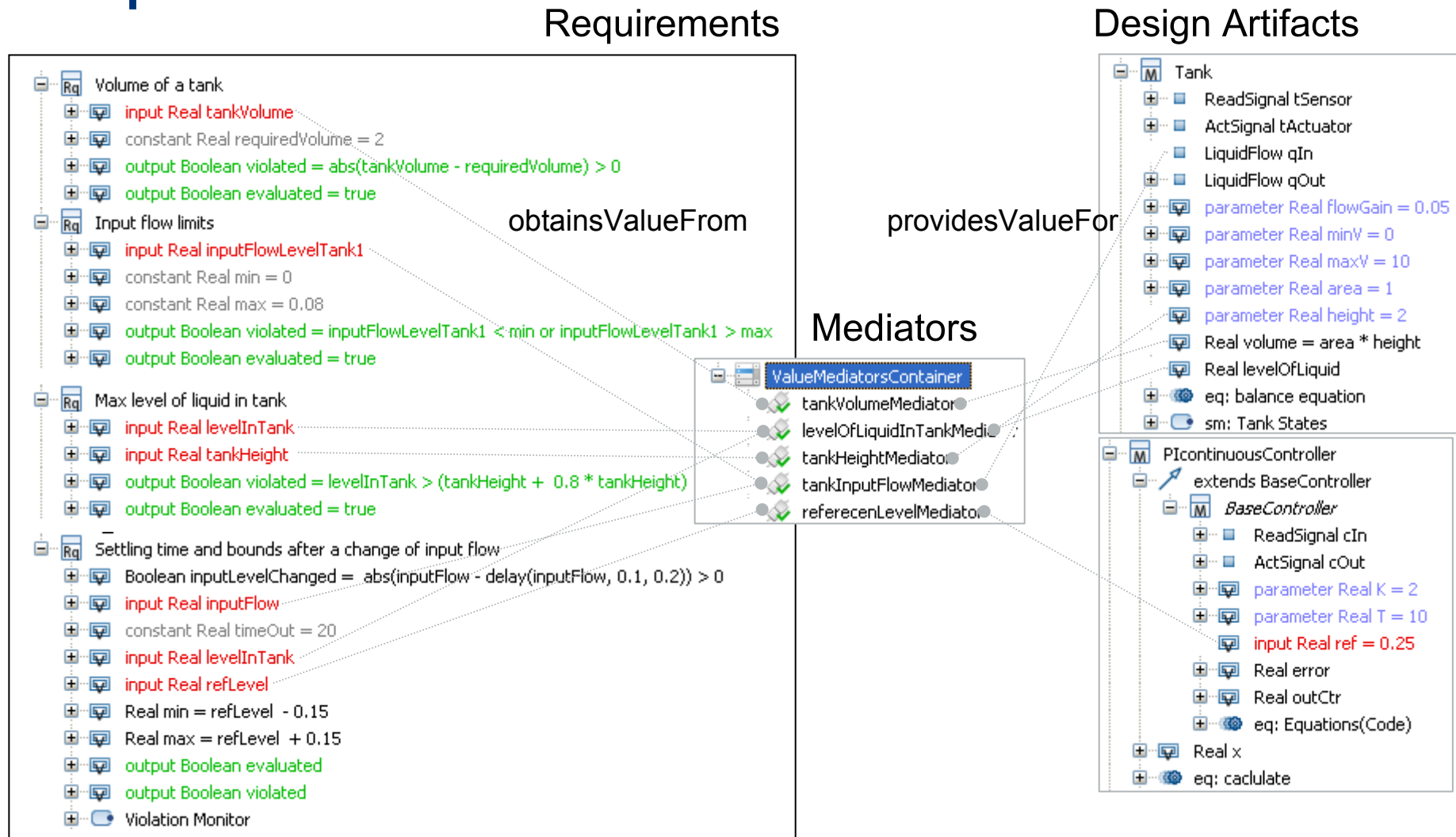
Handling of properties that are mentioned in multiple requirements

- ValueMediator concept is beneficial because it enables a consistent list of same properties that are reused in multiple requirements

Design Model Alternatives Handling

- ValueMediator concept is used to encapsulate the design model alternatives. Multiple design alternatives provide the value for the same ValueMediator.

Traceability Between Requirements and Design Properties



Finding the properties that are linked through mediators

Instantiates a class and enables the finding of different properties that are linked through mediators

