ModelicaML: Getting Started

Issue 1.6.8

February 10, 2013

Wladimir Schamai

EADS Innovation Works (Hamburg, Germany)

Linkoping University (Linkoping, Sweden)

Abstract: *This document provides a short introduction to ModelicaML and describes how to install the ModelicaML modeling and simulation environment.*

Table of Contents

Table of Figures

1 ModelicaML: Introduction

ModelicaML is a graphical modeling language and UML profile for the description of system architecture and system dynamic behavior. ModelicaML is based on an extended subset of the OMG Unified Modeling Language (UML) and is designed for Modelica code generation from graphical models such as state machines and activity diagrams, supporting hardware/software co-modeling and system requirement verification. ModelicaML is developed in collaboration between Linköping University and EADS Innovation Works, within the Open Source Modelica Consortium.

ModelicaML extends the graphical modeling capabilities of Modelica by providing standardized diagrams for presenting the composition, connection, inheritance, and behavior of system models. Moreover, ModelicaML incorporates methods for model-based verification of system requirements using simulations.
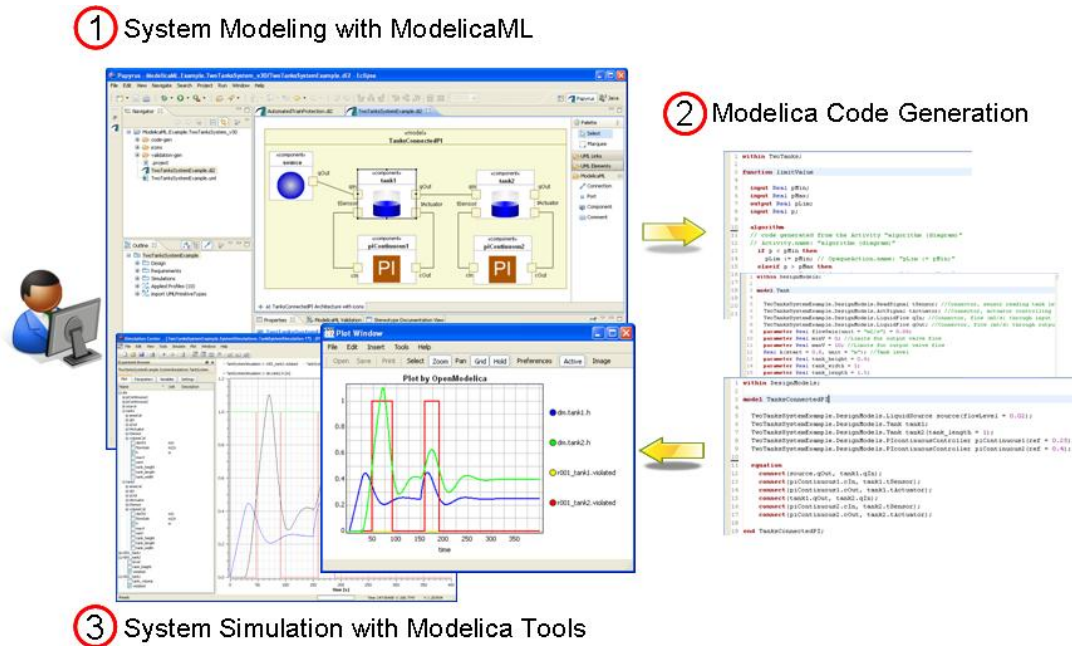
Figure 1: ModelicaML concept

The goal of ModelicaML is to enable an efficient and effective way to create, visualize and maintain combined UML and Modelica models. ModelicaML is defined as a graphical notation that facilitates different views (e.g., composition, inheritance, behavior) on system models. It is based on a subset of UML and reuses some concepts from SysML. ModelicaML is designed to generate Modelica code from graphical models. Since the ModelicaML profile is an extension of the UML meta-model it can be used as an extension for both UML and SysML[1].

UML/SysML provides the modeler with powerful descriptive constructs at the expense of sometimes loosely defined semantics that are marked as "semantic variation points" in the UML/SysML specifications. The intention in ModelicaML is to provide the modeler with powerful executable constructs and precise execution semantics that are based on the Modelica language. Therefore, ModelicaML uses a subset of UML, extends the UML meta-model (using the UML profiling mechanism) with new constructs in order to introduce missing Modelica concepts, and reuses some concepts from SysML. However, like UML and SysML, ModelicaML is mainly used as a graphical notation. ModelicaML models are eventually translated into Modelica code. Hence, the ModelicaML execution semantics are defined by the Modelica language and ultimately by a Modelica compiler that will translate the generated Modelica code into an executable form.

---

[1] SysML itself is also a UML Profile. All ModelicaML stereotypes that extend UML meta-classes are also applicable to the corresponding SysML elements.

Presently, the ModelicaML prototype is based on the following architecture:

- Papyrus [5] is used as modeling tool. It is extended by the ModelicaML profile and customized modeling tool features (e.g. dedicated toolbars, diagram selections, etc.).
- A ModelicaML model can be validated in order to check constraints and possible inconsistencies by using a validator plug-in which informs the modeler about inconsistencies or restriction violations. Note that the validation on the ModelicaML model does not replace the semantic analysis by a Modelica compiler.
- The ModelicaML code generator that generates Modelica code from the ModelicaML models is implemented using the Acceleo Eclipse plug-in [4], which follows the MDA approach and the model-to-text recommendations of the OMG.
- Finally, Modelica tools such as OpenModelica[7], Dymola[8] or MathModelica[9] are used to load the generated Modelica code and to simulate it.
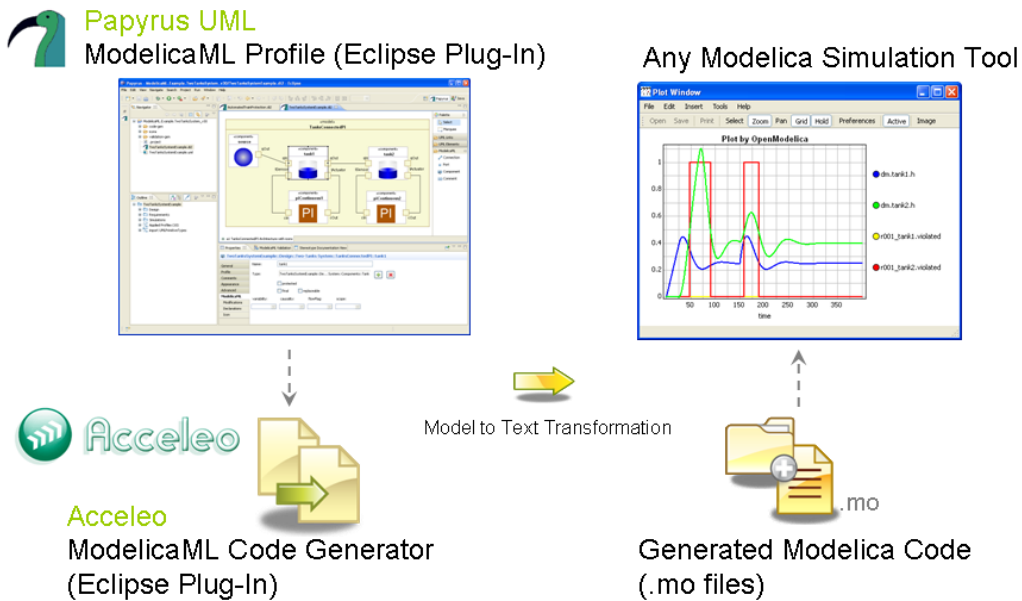
Figure 2: ModelicaML prototype architecture

Since the ModelicaML profile is an extension of the UML meta-model it can be used for both: modeling with the standard UML and with the SysML[2].

1.1 ModelicaML Notation Overview

The following UML diagrams can be used in ModelicaML for a graphical representation of model data:

- **Class Diagram:** for representing inheritance and class decomposition.
- **Composite Structure Diagram:** for representing class components and their interconnection. This diagram is similar to the *Modelica Connection Diagram*.
- **Activity Diagram:** for modeling of conditional equations or algorithm statements.
- **State Machine Diagram:** for modeling of state-based behavior.

---

[2] SysML itself is also a UML Profile. All stereotypes that extend UML meta-classes are applicable to the SysML as well.

Figure 3: ModelicaML notation

Textual requirements are represented using classes with additional attributes such as ID and text. It is possible to link requirements to other classes in order to establish traceability. Furthermore, it is possible to formalize requirements and to bind them to design models in order to evaluate requirements during system simulations. See the ModelicaML examples for more information.

1.2 ModelicaML GUI in Papyrus

ModelicaML plug-ins are implemented as extensions of the Papyrus UML tool. The extensions are:
- Diagram palette for creating ModelicaML elements on diagrams
- Model explorer actions for creating elements, diagrams and launching the validation and Modelica code generation, etc.

The following figures depict the graphical user interfaces.

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
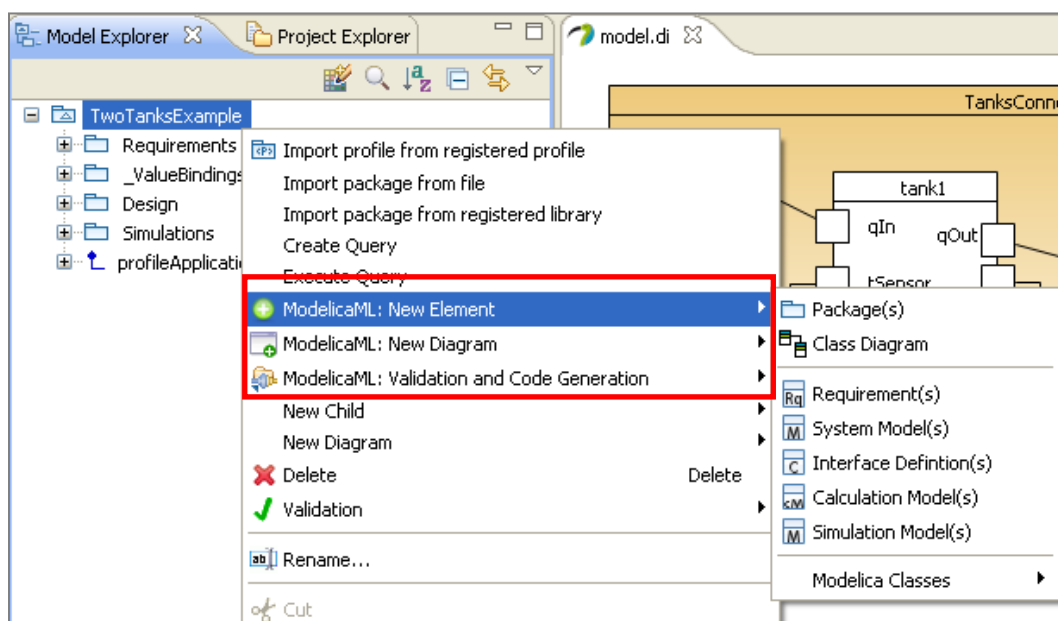
Figure 4: ModelicaML GUI in Papyrus

Figure 5: ModelicaML GUI actions in Papyrus

1.3   Validation of ModelicaML Models and Modelica Code Generation

A ModelicaML model can be validated by clicking on any model element (in the model browser or on diagrams and select "Validate ModelicaML model"). Violations or warnings are displayed in the ModelicaML Validation view (you will have to make it visible by going to window->Show view -> Other->ModelicaML->ModelicaML Validation view) and advise the modeler to correct the model. At the same time a name_validation_results.xml is created in the "**validation-gen**" folder of the project. The folder is created automatically and the files are overridden when regenerating.

Modelica code can be generated at any time independently of the result of the validation. Notice, the generator does not check for empty names or name-clashes between classes (e.g. two classes in a package that have the same name). Furthermore, this validation feature does not replace semantic validation as done by Modelica compilers.
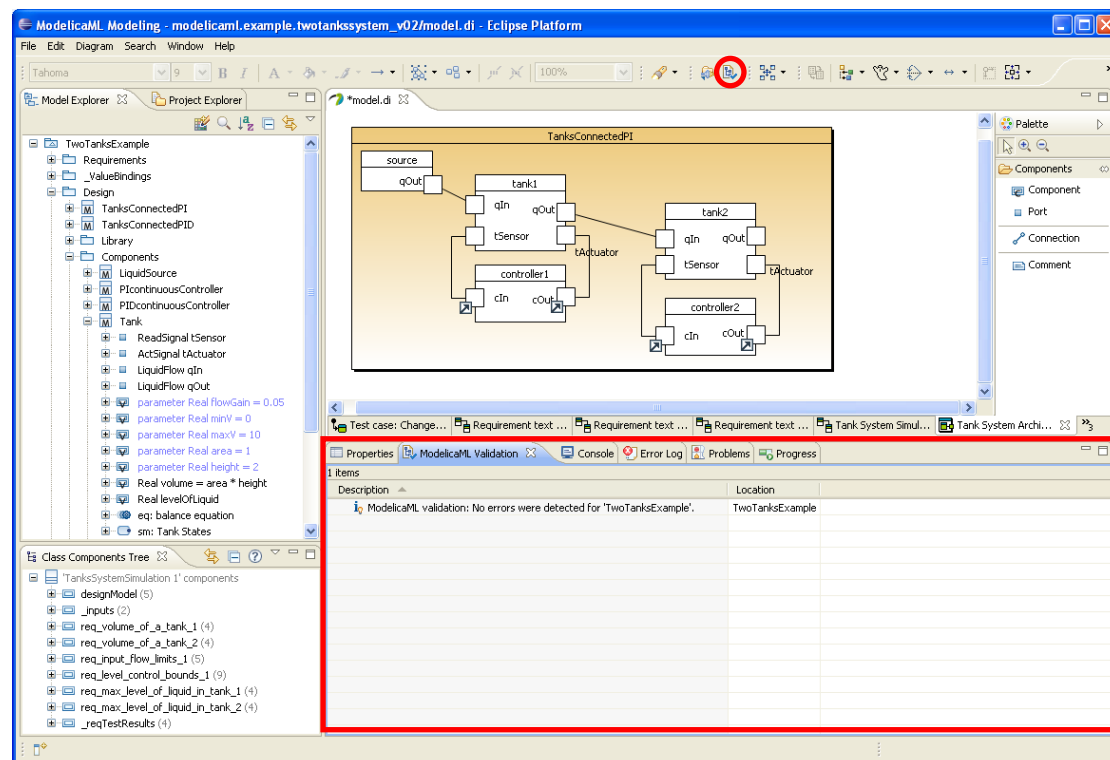
Figure 6: ModelicaML validation

1.4   Modelica Code Generation

The ModelicaML code generator generates textual Modelica code (folders, representing packages, and containing .mo files per class). The output folder is called "**code-gen**" (it will be created automatically if it does not exist) inside the defined project. A regeneration of code from a model will override all .mo files that exist in this folder (except for the use-code areas that are marked explicitly).

In order to launch the code generator click on any model element (in the model browser or on diagrams and select "Generate Modelica code from this ModelicaML model").

Figure 7: ModelicaML code generation

The code generator may modify the names of model elements as follows:
- It replaces all special characters and white spaces (if there are any)
- If a name starts with a number the code generator adds "_" in front of the number

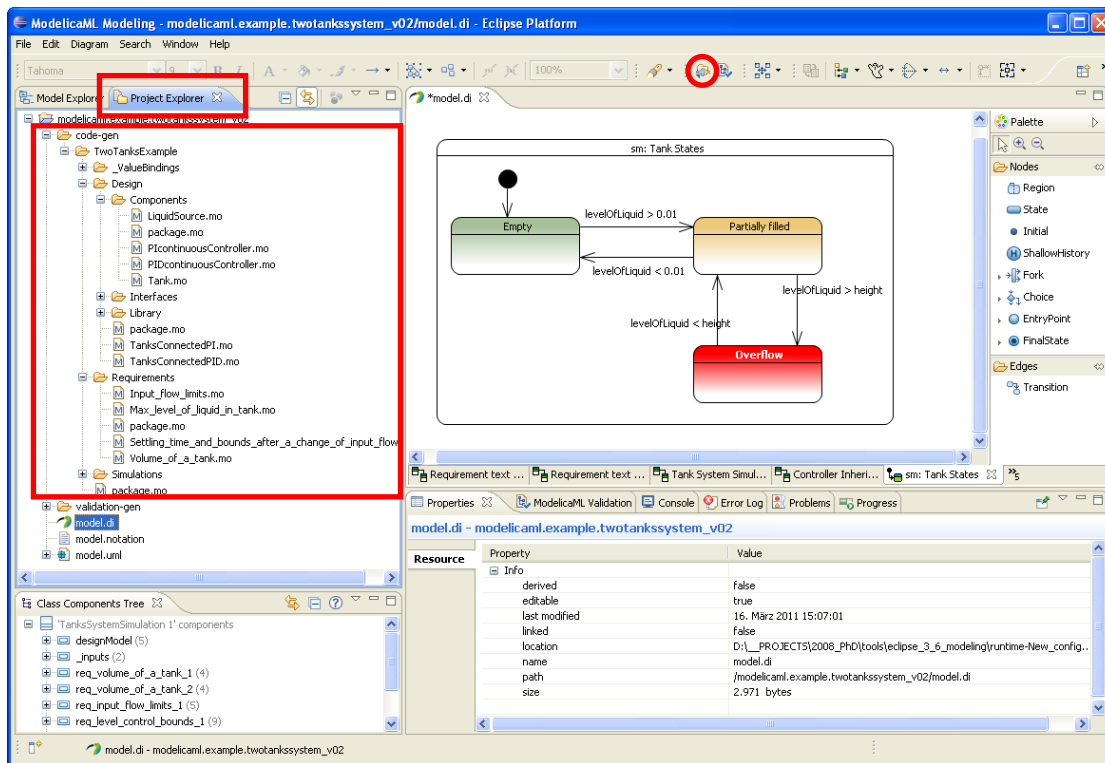## 1.5   Graphical Annotation

Some basic icon annotation (basic UML notation for a package, class and state machine) is generated from the ModelicaML model. However, no diagrams are translated into Modelica graphical annotation.

1.6   Using Existing Modelica Models in ModelicaML

It is possible to use existing Modelica models in ModelicaML. By convention, in order to enable it all Modelica models must be located in the **"code-sync" folder** (see Figure 8) inside your Eclipse ModelicaML project.
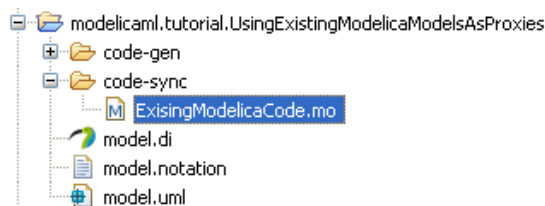
Figure 8: Convention for project folder structure

The central idea is that the created ModelicaML elements, called proxies, only reflect the Modelica models from the **"code-sync" folder.** This means that proxies should never be edited in ModelicaML but kept synchronized with the Modelica models from the "code-sync" folder. Therefore, whenever the Modelica models have change, the proxies should be synchronized as described in section 1.6.1.

There is the following restriction on the Modelica models to be imported:
- **The first level class must be a Modelica Package**

- The first level Modelica package should not have extends or import relations (those will not be imported)

Moreover, in the current version the following data is **not translated** (subject to future enhancements):
- **contrainedBy** relation between types
- **partial derivative function** relation between functions.

1.6.1 How to Use It

Alternative 1: To create new proxies or synchronize the Modelica models from the "code-sync" folder with the existing proxies click on the "Synchronize Modelica Model Proxies" from the toolbar (see Figure 9) and follow the dialog instructions. For small or mid-size models the synchronization takes seconds. For large models, such as entire MSL, the synchronization may take minutes or hours.
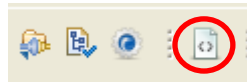
Figure 9: Proxy Synchronization toolbar button

Alternative 2: In addition, there is an alternative graphical Eclipse view (see Figure 10) that provides a tree-based overview of the loaded Modelica models, and has separate actions for loading and synchronizing proxies. The tree-items are decorated with colored text-style and error or warning overlay images. These are useful features; however, the current implementation is not yet efficient in coping with large models. It is not recommended to use the graphical Eclipse view (from Figure 10) when synchronizing large models such as the entire Modelica Standard Library (in such a case **Alternative 1** from above should be used).
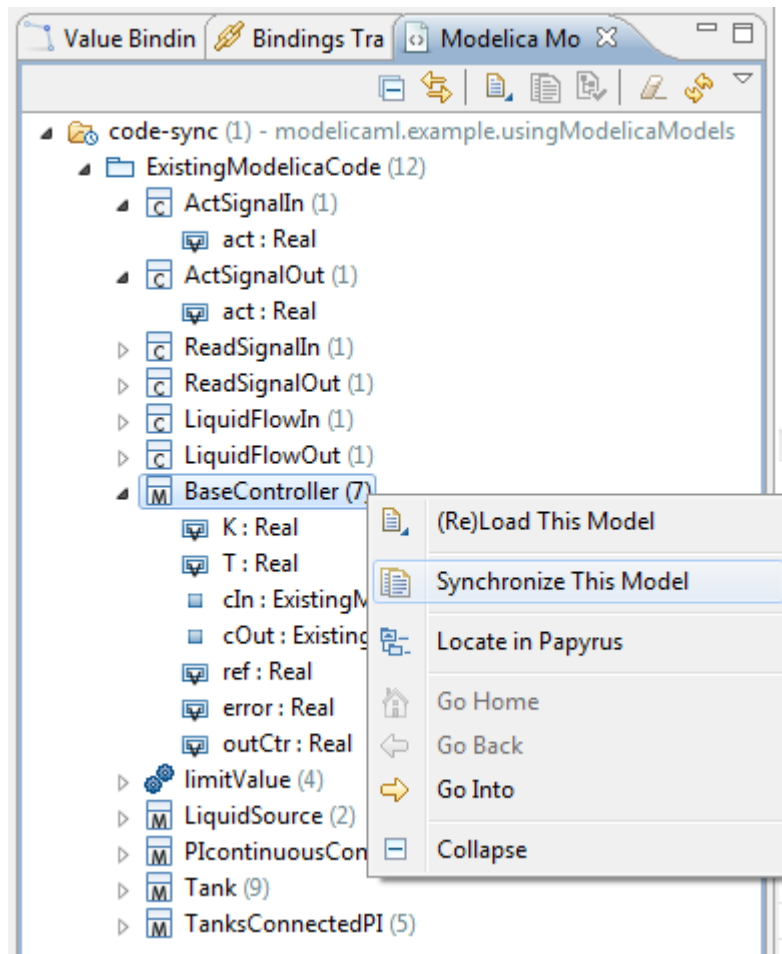
Figure 10: Proxy Synchronization View

Result: The proxies are created in the ModelicaML model as additional root nodes (see Figure 11) and can be referenced by other ModelicaML elements.
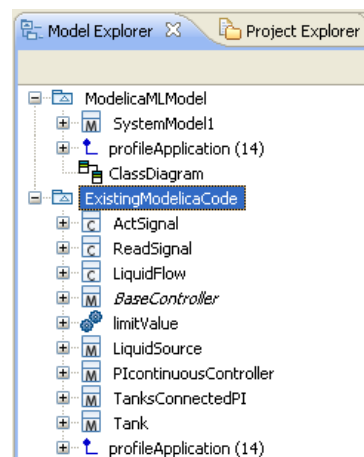
Figure 11: Example of created proxies

Note that no Modelica code is generated from proxies. Instead, f**or simulation** both, the generated code **(from the "code-gen" folder)** and the referenced Modelica models **(from the "code-sync" folder)** must be loaded into the Modelica compiler. Moreover, if classes from the Modelica Standard Library (MSL) are used then the **MSL should also be loaded** in the Modelica compiler before simulating a model.

1.7 Using Modelica Standard Library (MSL) in ModelicaML

Modelica Standard Library is provided as proxies (see section 1.6) within the ModelicaML profile. The following figure illustrates how to import a registered library in Papyrus. To import the MSL right-click on your top-level model, select "Import package from registered library" and select "Modelica Standard Library".
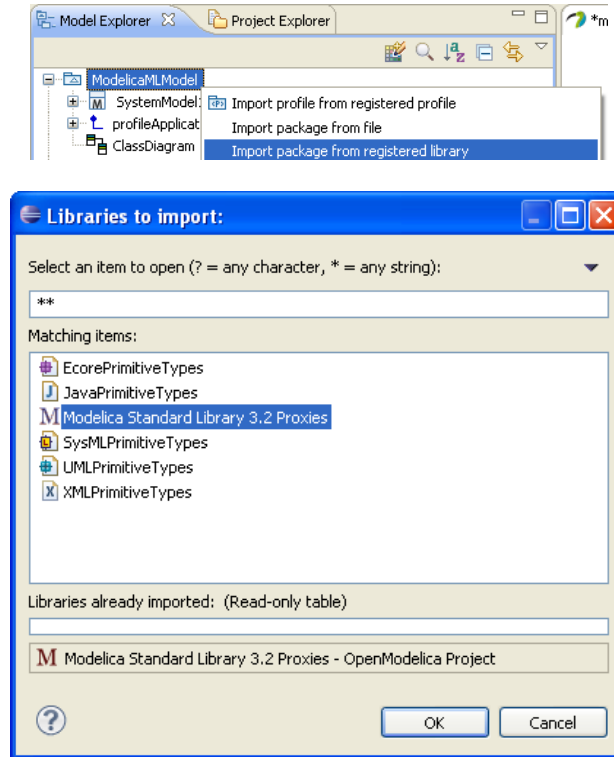

Figure 12: Importing a registered library in Papyrus

1.8 Subset of the UML Used in ModelicaML

ModelicaML uses only a subset of the UML as basis meta-model. The following UML meta-classes (their interrelations are implicit and are not listed here) from [2] are used in ModelicaML models and are eventually translated into Modelica code. Most of them are extended by the ModelicaML Profile in order to facilitate the capturing of required Modelica constructs. Some are not extended and are used as defined by the UML (e.g. InitialNode, ControlFlow, Enumeration, Comment, etc.). The numbers in the title-rows correspond to the chapter-numbers in [2].

7 Classes		11 Actions		13 Common Behaviors
Class		CallBehaviorAction		FunctionBehavior
Comment		OpaqueAction		OpaqueBehavior
Constraint				
Enumeration		12 Activities		15 State Machines
Generalization		Activity		Pseudostate
Package		ControlFlow		Region (parallel states)
Parameter		DecisionNode		State
Dependency		MergeNode		StateMachine
		InitialNode		Transition
9 Composite Structures				
Class				
Connector				
Parameter				
Port				

Property		

1.8.1 Subset of UML2 Activity Concepts Supported in ModelicaML

A subset of the UML Activity is used in ModelicaML to model conditional equations or algorithm statements. The only constructs inside an Activity (with respective stereotype applied indicating that it is an equations or algorithm section) that are supported today are:
- InitialNode (exactly one),
- ControlFlows,
- OpaqueActions (for capturing the assignments or equations),
- CallBehaviorAction (for hierarchical modeling),
- MergeNode and
- DecisionNode (with if/when stereotypes).

Note that the semantic of this sub-set of UML activity is not based on the token-principle. This notation used for modeling conditional constructs, such as Modelica `if` or `when`.

1.8.2 Subset of UML2 State Machines Concepts Supported in ModelicaML

UML defines two types of state machines: Behavior state machines and protocol state machines. Behavior state machines are used to model parts of class behavior. ModelicaML state machines are derived from UML behavior state machines. Compared to behavior state machines, protocol state machines are limited in terms of expressiveness and are tailored to the need to express protocols or to define the lifecycle of objects. Since this is not the main intended area of application for ModelicaML, the protocol state machines are not taken into account. Consequently, none of the chapters of the UML specification that address the protocol state machines are considered.

1.8.2.1 Support of Graphical Notation

The graphical notation for ModelicaML state machines is based on the notation for UML behavior state machines. All modeling constructs are supported except *SendSignalActions* or other actions (see [2], p.578) which are **not** supported graphically. Instead the modeler can capture required behavior in the behavior bodies (e.g., entry/do/exit behavior of states).

1.8.2.2 Supported Subset of UML State Machine Concepts

In the following a list of UML state machine concepts[3] is presented which are not supported in ModelicaML or which are supported in a different way:

- **15.3.2 FinalState (from BehaviorStateMachines)**: Supported in ModelicaML. However, no explicit completion events are generated if regions of a composite state have reached the FinalStates (see UML[2] p.574, see also comments for "*Completion transitions and completion events*" below). Furthermore, no termination of the context object (i.e. class owning the state machine) is implied if all regions of a composite state reach their FinalStates (see also comments for terminate state below).
- **13.3.6 CallEvent** (from Communications):
  - *CallEvent* implies a call of a class operation. This is not applicable to Modelica since Modelica does not support the class-method concept that can be found in object-oriented languages such as Java or C++.
- **15.3.8 Pseudostate** (from BehaviorStateMachines), **15.3.9 PseudostateKind** (from BehaviorStateMachines):
  - *DeepHistory* is **not** supported in ModelicaML.
  - *Terminate:* Reaching a terminate state in ModelicaML does not imply termination of the context object (i.e. class owning the state machine). Reaching a terminate state implies the deactivation of state machine.
- **15.3.11 State** (from BehaviorStateMachines, ProtocolStateMachines):
  - *Deferred events* and *State redefinition* are **not** supported in ModelicaML.

---

[3] The chapter numbers correspond to the UML[2] specification chapter numbers.

- **15.3.12 StateMachine** (from BehaviorStateMachines):
  - Priority scheme for conflicting transitions that are at different hierarchy levels is different from UML. UML defines that "*The priorities of conflicting transitions are based on their relative position in the state hierarchy. By definition, a transition originating from a sub state has higher priority than a conflicting transition originating from any of its containing states.*" UML [2], p. 567). The ModelicaML priority scheme for conflicting transitions is different.
  - *StateMachine extension* is **not** supported in ModelicaML.
- **15.3.14 Transition** (from BehaviorStateMachines):
  - *Internal (and local) transition* are **not** supported in ModelicaML. Only *external transitions* are supported.
  - *Completion transitions and completion events* as defined in UML (see p. 574) are **not** supported in ModelicaML. The reason for this is that for ModelicaML state machines a different priority scheme for conflicting transitions (that are at different hierarchy levels) is used. The same behavior can be expressed in ModelicaML using `isInState(…)` macro or joins).
  - *Deferred triggers* and *Transition redefinition* are **not** supported in ModelicaML.
- **15.3.15 TransitionKind** (from BehaviorStateMachines): Only kind = external is supported in ModelicaML.

1.8.3   Predefined Macros

The following macros can be used inside transition guard definition or inside the bodies of entry/do/exit state-actions or transition effects.

1.8.3.1   Macros inside Transition Guards

1.8.3.1.1   AFTER - Macro
- Semantic: AFTER() macro is dedicated to state transitions. The meaning is that after a state is entered and the local state time exceeds the given number this part of the transition-guard evaluates to true.
- Syntax: `AFTER(expression)`
- Example: `AFTER(23)`
- Expanded to: `time – `*`state_path`*`.timeAtActivation > 23`

1.8.3.1.2   CHANGE - Macro
- Semantic: CHANGE() macro has the same meaning as the Modelica change() function.
- Renamed to: `change(`*`variable_name`*`)`

1.8.3.1.3   Absolute Value - Macro
- Semantic: Absolute value macro has the same meaning as the Modelica abs() function.
- Syntax: `|variable_name|`
- Expanded to: `abs(variable name)`

1.8.3.2   Macros inside Action Bodies

1.8.3.3   GEN_CHANGE - Macro
- Semantic: GEN_CHANGE() is used to negate a Boolean variable.
- Syntax: `GEN_CHANGE(`*`name_of_the_boolean_variable`*`)`
- Example: `GEN_CHANGE(var1)`
- Expanded to: *`name_of_the_boolean_variable`*` := not `*`name_of_the_boolean_variable`*

1.9 Limitations With Regard To Modelica

Since ModelicaML is based on the UML meta-model some limitations exist with regard to the Modelica code to be generated or imported. These limitations are particularly important when importing existing Modelica code (e.g. Modelica Standard Library) into UML-based (e.g. ModelicaML) models:

- Graphical Decomposition of Ports (Instances of Connector Classes): In UML it is not possible to represent Ports decomposition (in ModelicaML these represent instances of connector classes) graphically.
- Short-Class-Definition: Modelica short-class-definition is not possible in ModelicaML. The modeler should use inheritance constructs instead.

2 Installation

The ModelicaML modeling and code generation environment is implemented using the Eclipse technology and includes the following plug-ins:

- Papyrus [5] as UML-based modeling tool (for ModelicaML modeling)
- Acceleo [4] for model to text transformation (for Modelica code generation)

Acceleo[4] became Eclipse Projects and is now based on the OMG model-to-text standard specification. For the future, it is planned to migrate the current environment from Acceleo 2.8 to Acceleo 3.x version.

The current ModelicaML Eclipse implementation has been tested for the following configuration:

- **Eclipse (Juno), Papyrus 0.9, Acceleo 2.8, Xtext 2.3 version**

2.1 Install OpenModelica Compiler (OMC)

Follow the instruction from http://www.openmodelica.org/index.php/download to install OpenModelica compiler (OMC) that is used for simulating ModelicaML models. **It should be installed before Eclipse is installed.**

2.2 Download Eclipse Modeling Tools Package (Eclipse Juno)

Description: Eclipse framework is used by ModelicaML modeling and code generation tools. Provider website: www.eclipse.org/downloads/

Procedure:

- Download, extract and start **Eclipse Modeling Tools**
- Update the network connection properties (if required) so other plug-ins can be installed.
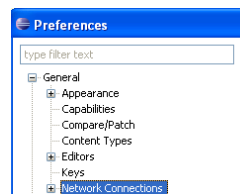

Figure 13: Eclipse Network Connections

2.3 Install Papyrus (UML Modeling Tool)

Description: Papyrus UML Modeler is used to create ModelicaML models.

Procedure:

- In Eclipse click Help -> Install Modeling Components -> Select Papyrus -> Install
- Update the network connection properties (if required) so other plug-ins can be installed.
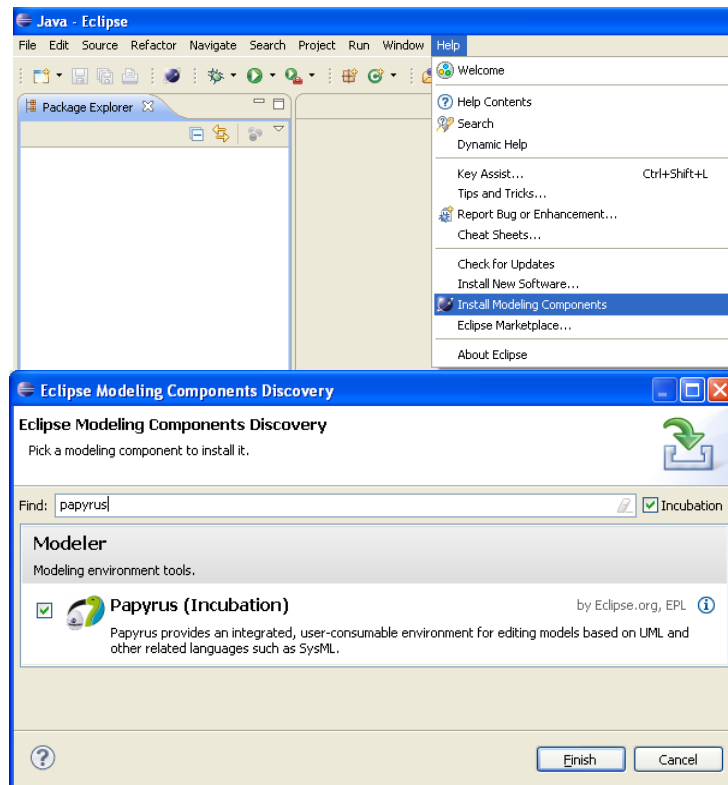
---

[4] http://www.eclipse.org/modeling/m2t/?project=acceleo

Figure 14: Install Papyrus

Provider website: http://www.eclipse.org/modeling/mdt/papyrus/
Eclipse Update Site: Part of the Eclipse Modeling Package

2.3.1   Update Papyrus

It is recommended to update the Papyrus release that is shipped with the Eclipse Modeling
Package. The Eclipse update site for the Papyrus nightly updates is:
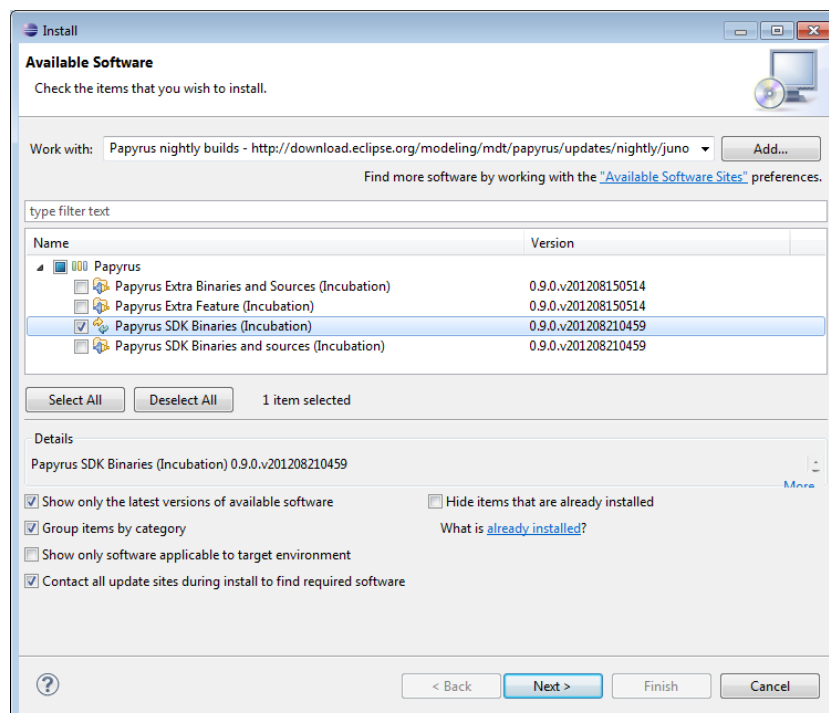**http://download.eclipse.org/modeling/mdt/papyrus/updates/nightly/juno**

Figure 15: Update Papyrus

2.4 Install Xtext (ModelicaML Action Code Editing)

Description: Xtext - Language Development Framework.
Provider website: http://www.eclipse.org/Xtext/
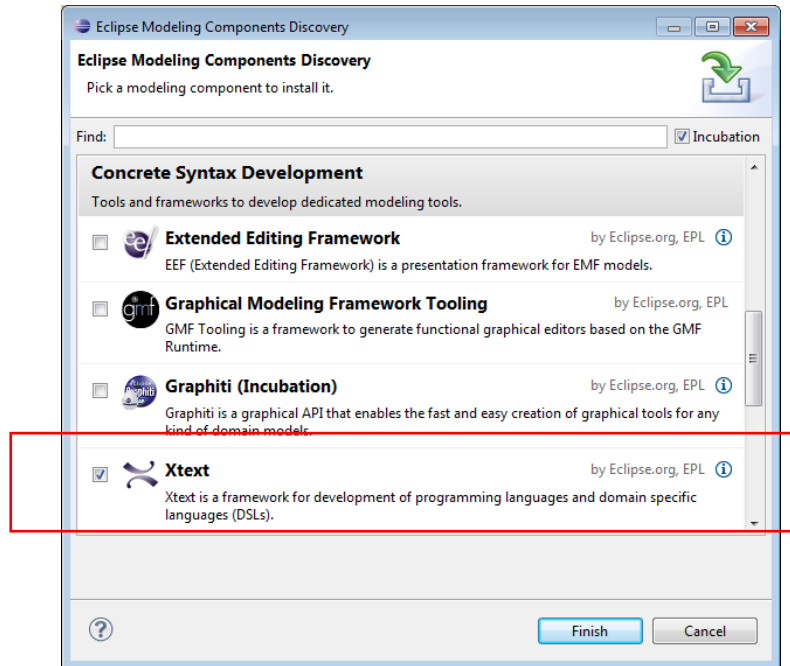Eclipse Update Site: Part of Eclipse Modeling Package

Figure 16: Install Xtext

2.5 Install Acceleo 2.8 (Modelica Code Generation)

Description: Acceleo technology is used for the generating of Modelica code from ModelicaML models.
Provider website: http://www.acceleo.org/pages/home/en
Eclipse Update Site: http://www.acceleo.org/update/

Note, the current version is based on Acceleo 2.8. Do not install Acceleo 3.0.

Procedure:
- In Eclipse go to Help -> Install New Software
- Add the Update-Site
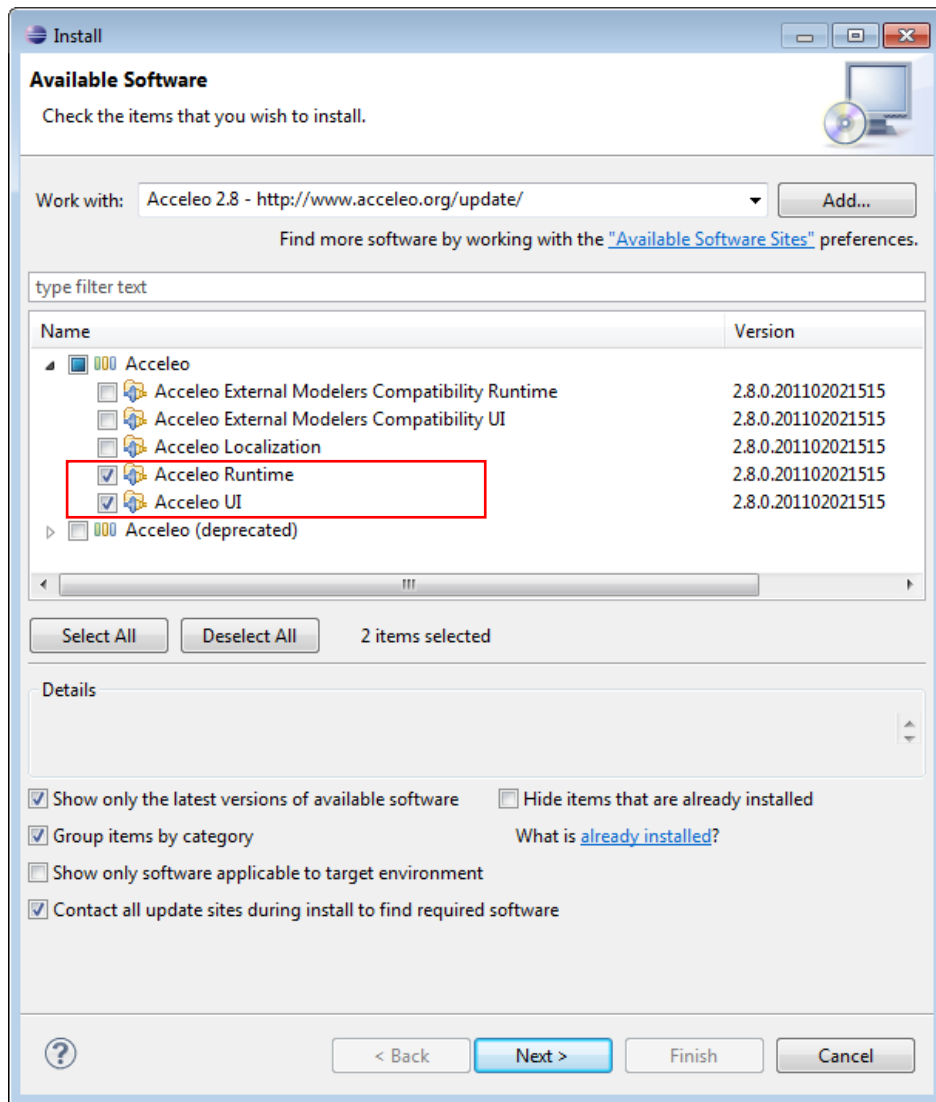- Install Acceleo Components (see screenshot)

Figure 17: Install Acceleo

2.6 Install Modelica Development Tooling

Description: MDT is used for viewing and editing of Modelica code within Eclipse environment and for connecting ModelicaML tool to OpenModelica Compiler (OMC) for model validation and simulation.

Provider Site: http://www.openmodelica.org/index.php/developer/tools/133
Eclipse Update Site: http://www.ida.liu.se/~pelab/modelica/OpenModelica/MDT/

Procedure:
- In Eclipse go to Help -> Install New Software
- Add the Update-Site
- Install MDT (see screenshot)
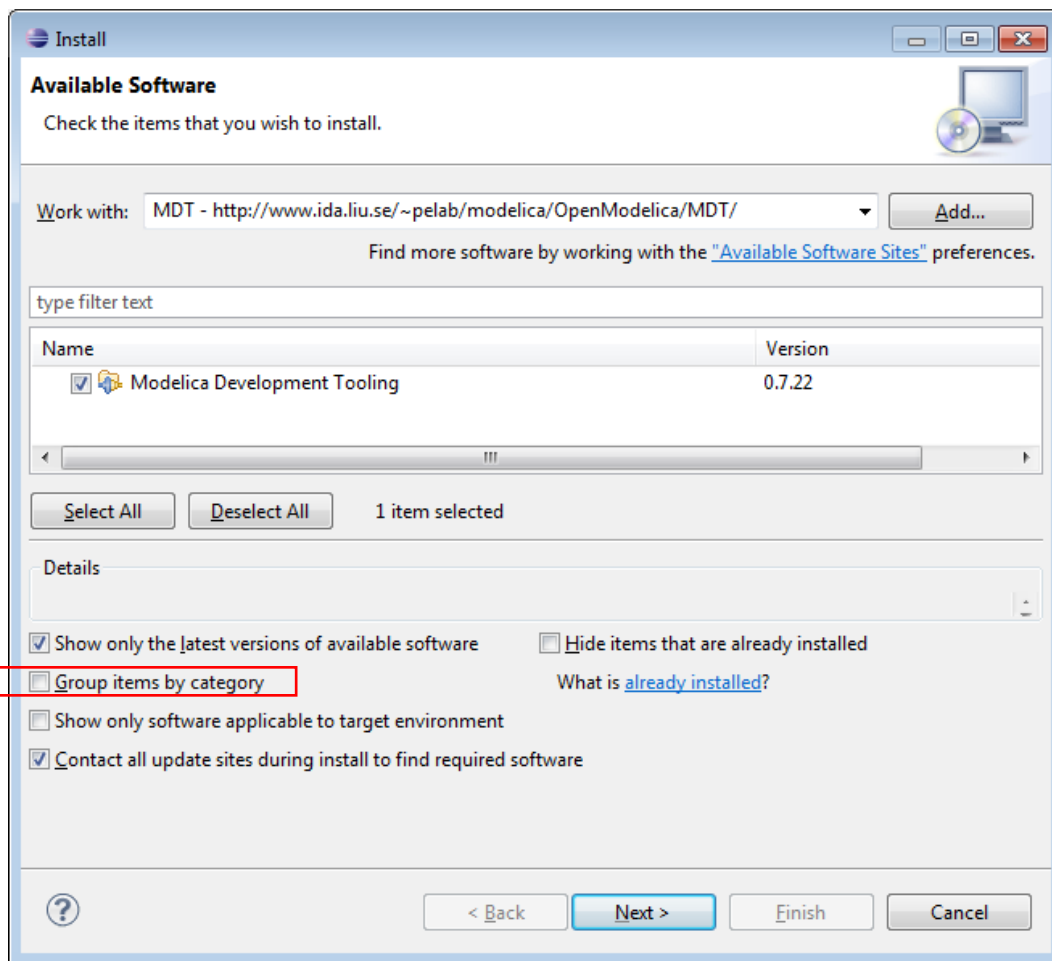
Figure 18: Install Modeling Development Tooling (MDT)

2.7 Install ModelicaML

Provider Site: http://www.openmodelica.org/modelicaml
Update site:
        http://www.ida.liu.se/~pelab/modelica/OpenModelica/MDT/ModelicaML/update/juno/
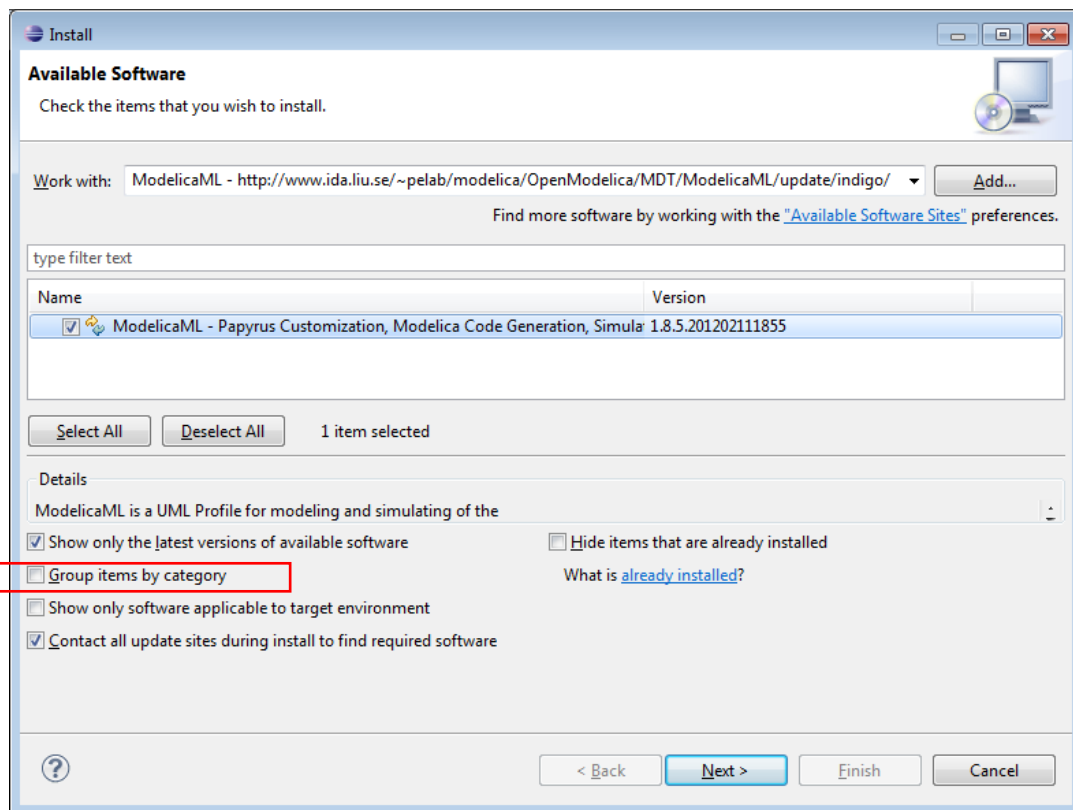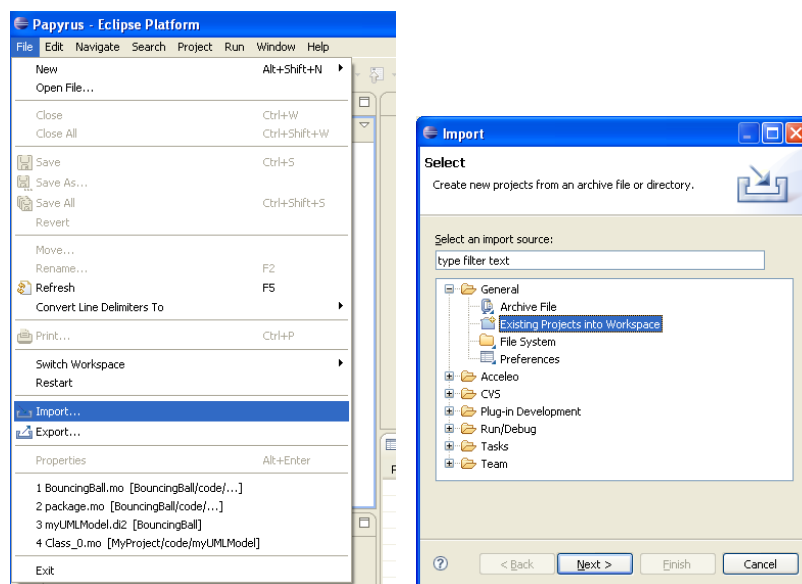
Figure 19: Install ModelicaML for Papyrus

2.8   Import an Example Projects

- Download the example project from the ModelicaML website [11]
- Go to File -> Import Existing Project into Workspace
- Select the example project
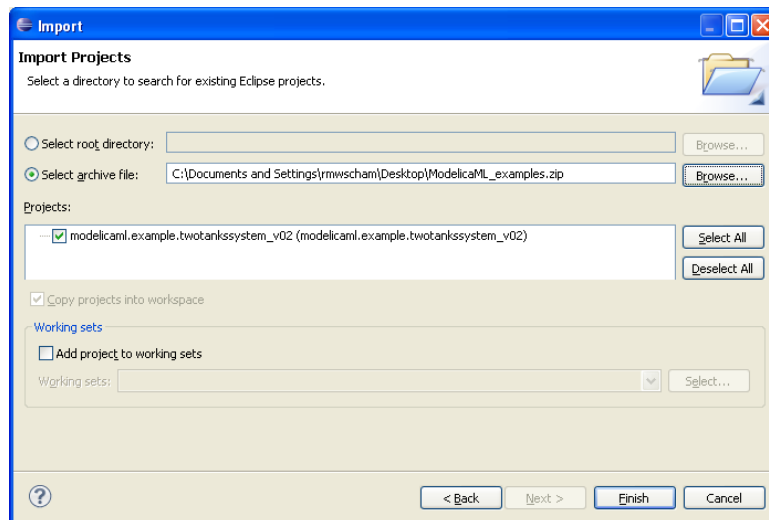- Select the option "copy projects into workspace"

Figure 20: Import ModelicaML examples

3 Customization

3.1 Eclipse.ini

Components of ModelicaML typically require more memory allocated to Eclipse as defined by default. In order to avoid "out of memory" messages you should modify your eclipse.ini file in order to allocate more memory to eclipse. Add to your eclips.ini the following lines (adopt if needed):

```
openFile
--launcher.XXMaxPermSize
1024M
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
1024m
--launcher.defaultAction
openFile
-vmargs
-Dosgi.requiredJavaVersion=1.5
-Dhelp.lucene.tokenizer=standard
-XX:MaxPermSize=2048m
-Xms512m
-Xmx2048m
```

3.2 Eclipse ModelicaML Perspective

ModelicaML defines no perspective layout by default. You can define the layout by first selecting all views you want to see (go to "*Window-> Show View -> ModelicaML*" and select the views including "*Project Explorer*" and "*Model Explorer*"). Then you can drag, move and drop the views in order to define your own workbench layout. A typical ModelicaML perspective layout could look like in the figure below.
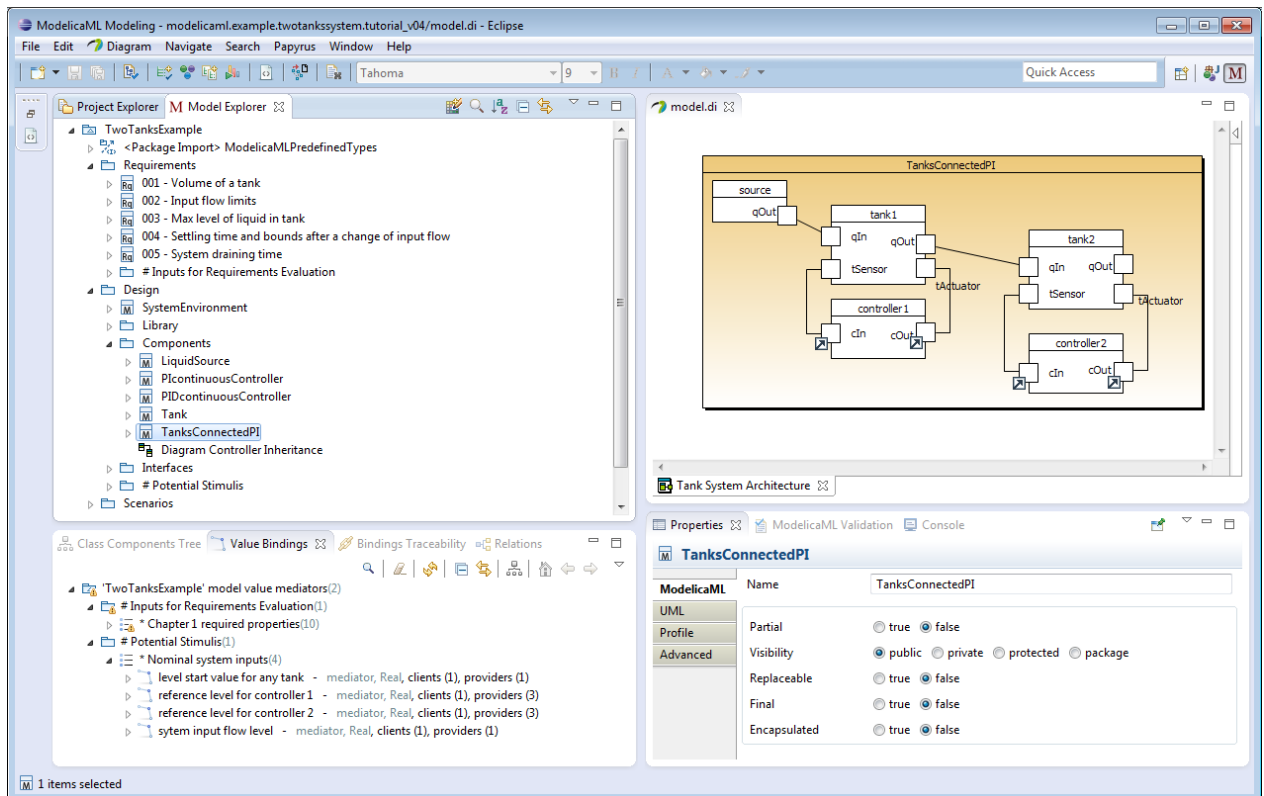
Figure 21: Example ModelicaML Perspective layout

Moreover, you can customize the Eclipse toolbar and menus by first switching to "*ModelicaML Modeling*" perspective and then got to "*Window -> Customize Perspective*". A customization could look like this:
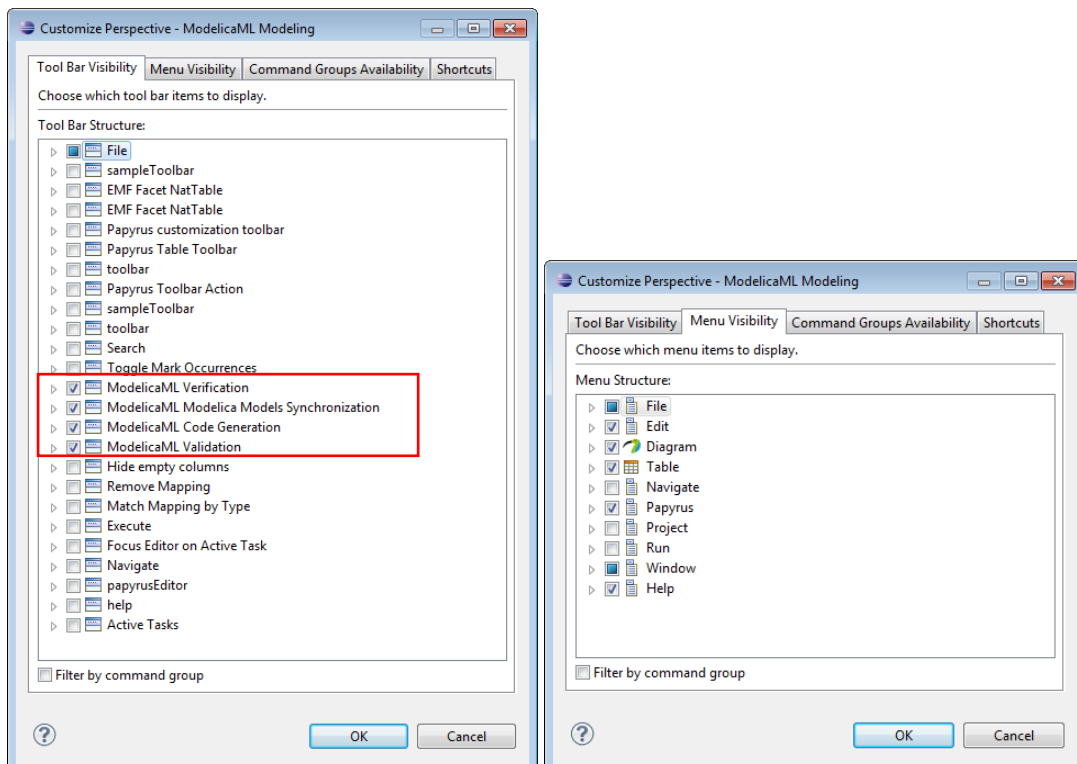

Figure 22: Example Perspective Customization

3.3   Preference Pages

3.3.1   ModelicaML and Papyrus Preference Pages

There ModelicaML and Papyrus and preference pages that you can use for the definition of default settings.
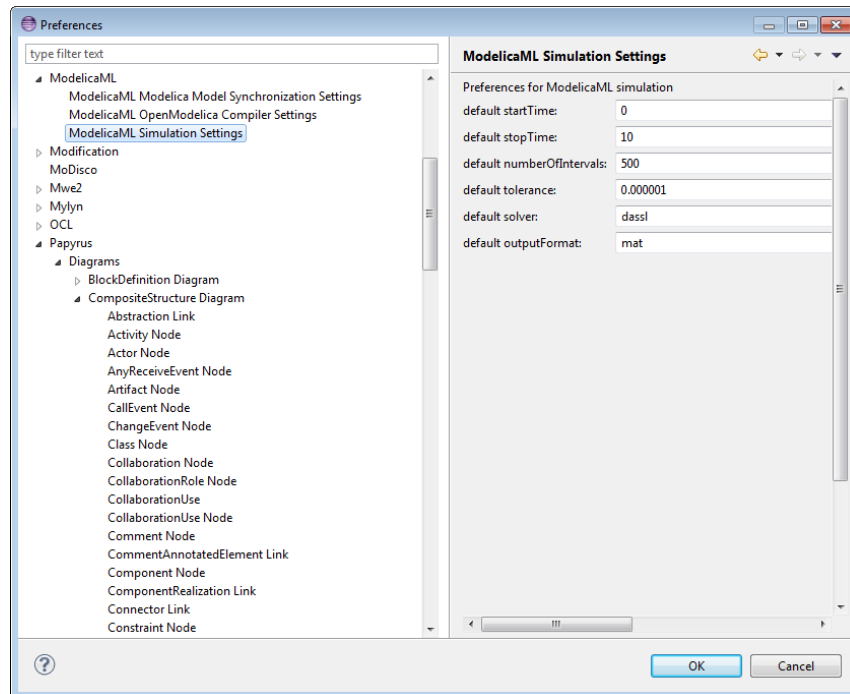

Figure 23: ModelicaML and Papyrus Preference Pages

3.3.2   Model Validation Customization

ModelicaML validation is based on EMF Validation Framework [12]. Other plugins may contribute (constraints) rules for validating the same models elements as are contained in ModelicaML models. In order to validate your model only according to the ModelicaML validation rules you should adopt the preferences as shown in the figure below.
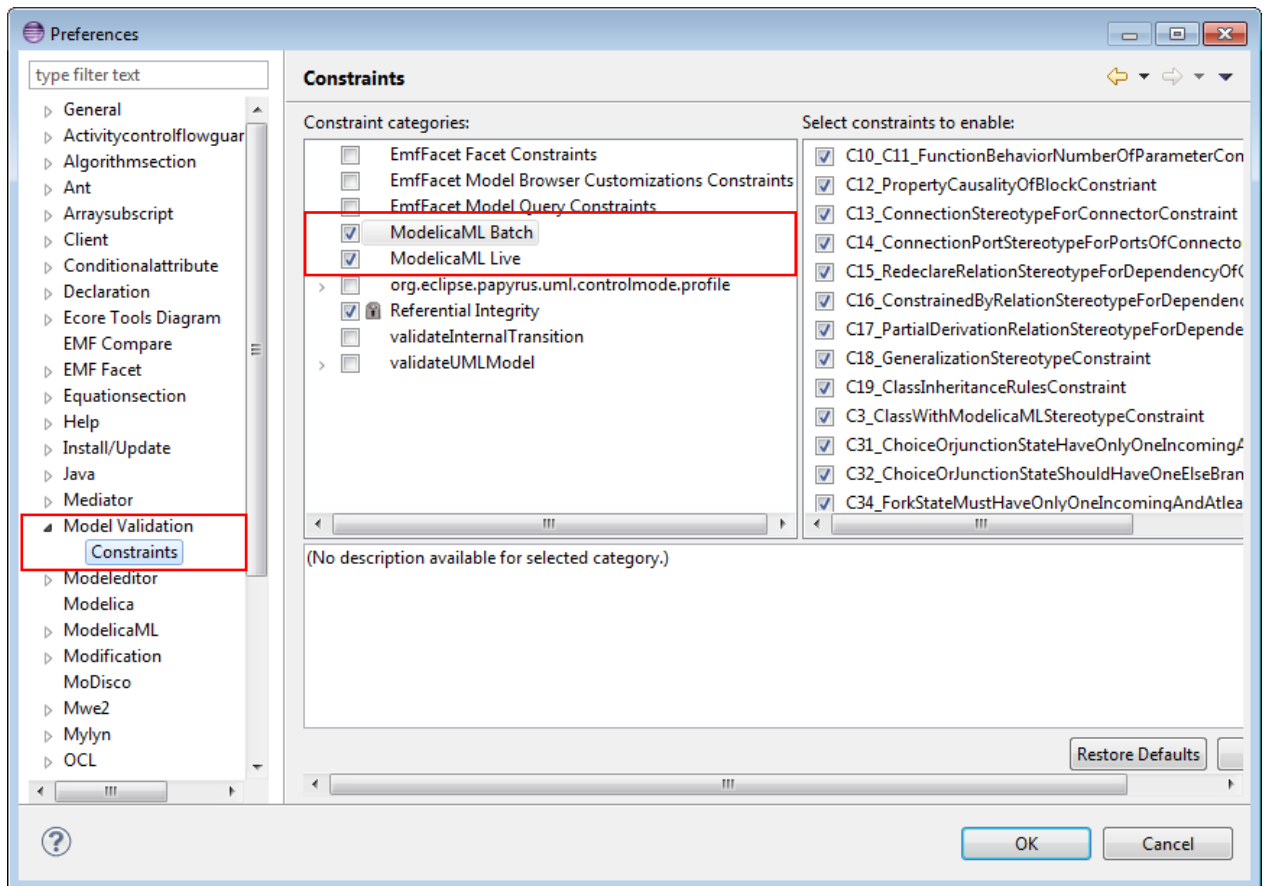
Figure 24: Model Validation Preference Pages

3.4 Papyrus Customization

In case the ModelicaML (with "ModelicaML Modeling" perspective) is not loaded by default on the on top in Papyrus you should click on "Load Browser Custiomization" and select ModelicaML Customization to be on top as depicted in the figure below.
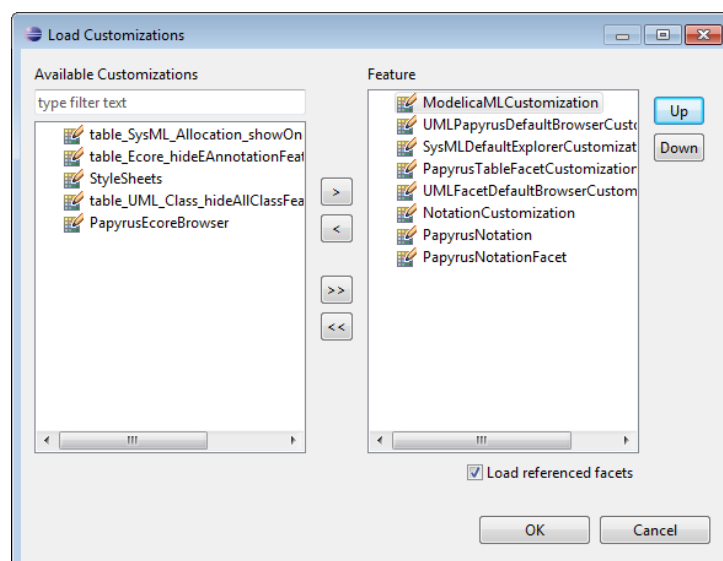

Figure 25: Loading ModelicaML Model Explorer Customization

3.5   Papyrus Diagram Palette Customization

Each diagram in Papyrus can have multiple palettes. If you only want ModelicaML palettes to appear on your diagram you should de-select other palettes are shown on the figure below.
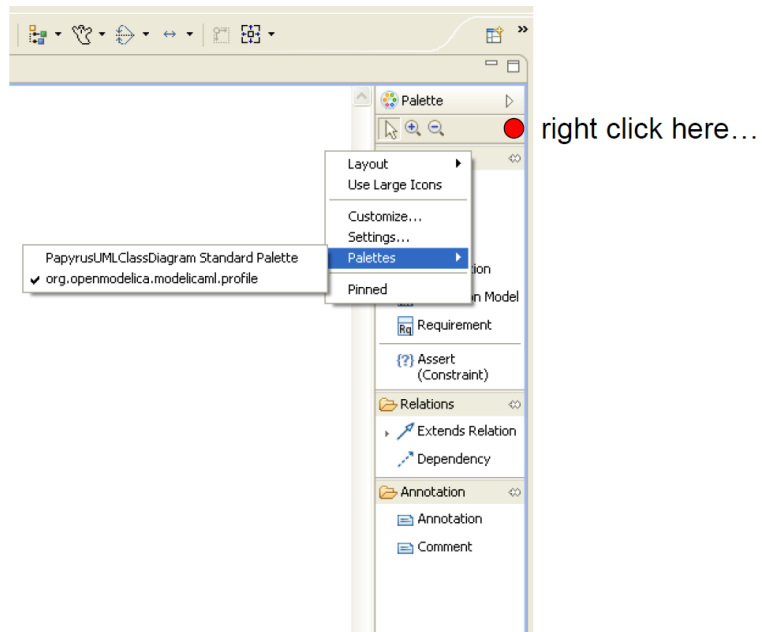

Figure 26: Papyrus Diagram Palettes

References

[1] Modelica Association. Modelica: A Unified Object-Oriented Language for Physical Systems Modeling: Language Specification Version 3.0, Sept 2007. www.modelica.org

[2] OMG. OMG Unified Modeling Language ^{TM} (OMG UML). Superstructure Version 2.2, February 2009.

[3] OMG. OMG Systems Modeling Language (OMG SysML™), Version 1.1, November 2008.

[4] Acceleo, Eclipse Plug-In. www.acceleo.org/pages/home/en

[5] Papyrus, www.eclipse.org/modeling/mdt/papyrus/

[6] Schamai W. Modelica Modeling Language (ModelicaML) A UML Profile for Modelica, Technical Report 2009:5, EADS IW, Germany, Linkoping University, Sweden, www.ep.liu.se, 2009

[7] The OpenModelica Project www.openmodelica.org

[8] Dymola (Dynamic Modeling Laboratory), Dynamism. www.dymola.com

[9] MathModelica, http://www.mathcore.com

[10] Acceleo, Eclipse Plug-In. www.acceleo.org/pages/home/en

[11] ModelicaML website: www.openmodelica.org/modelicaml

[12] Eclipse Modeling Framework Project (EMF) http://www.eclipse.org/modeling/emf/?project=validation