ModelicaML Tutorial

MODPROD 2011

Wladmir Schamai

(EADS Innovation Works, Linkoping University)
ModelicaML
A UML Profile for Modelica

Introduction
ModelicaML: History

• 2007: ModelicaML work started at PELAB
  – This version was based on the SysML

• Since Jan. 2009: OMG SysML / Modelica Integration WG (ongoing)
  – Main target: Integration of Modelica (executable) models into (descriptive) SysML models

• Since Jan. 2009: ModelicaML – Next Generation
  – Is a research activity as part of a PhD
  – Contributes to the OMG SysML / Modelica Integration WG
  – Is based on the “pure” UML and implemented as a UML profile
  – Main objective: Executable graphical modeling language for hybrid system dynamics modeling and simulation
Introduction and Motivation

• Graphical modeling promises to be more effective and efficient, regarding editing, human reader perception of models, and maintaining models compared to a traditional textual representation
  – UML/SysML provide a comprehensive set of graphical notations
  – Modelica is defined as textual syntax, the graphical notation is limited to domain specific notations and one diagram type

• Existing UML/SysML formalisms are typically translated into (and limited to) the time-discrete or event-based simulation of a system or software.
  – This is a critical limitation with regard to physical systems simulation
  – This limitation disappears when Modelica comes into the play

• **Motivation**: Integration of UML/SysML in order to combine
  – The descriptive power of UML/SysML notations
  – The simulation power of Modelica language (logical and physical system simulation)
Modelica: Graphical Modeling

- Domain-specific icons on one diagram type: *Connection Diagram* (hierarchical modeling of system structure and the interconnection of components)
- No graphical representation for behavior (equations, assignments, conditionals)
ModelicaML: Today

- ModelicaML is a UML Profile
  - Applicable to pure UML or to other UML profiles, e.g. SysML

- ModelicaML
  - Enables capturing of all constructs of the Modelica language
  - Defines graphical concrete syntax (diagrams notation) for representing Modelica constructs
  - Includes graphical formalisms (e.g. State Machines, Activities, Requirements)
    - Which do not exist in Modelica language
    - Which can be translated into executable Modelica code
  - Is defined towards generation of executable Modelica code
ModelicaML: Technology

1. System Modeling with ModelicaML

2. Modelica Code Generation

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)
ModelicaML

Graphical Notation
ModelicaML: Graphical Notation

Structure

Requirements

Behavior

- ModelicaML: Graphical Notation
- Structure
- Requirements
- Behavior

- 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

- **State Machine**: «modelicaStateMachine» player modes
- **Composite State**: on
- **Start**: Set discharge rate /do
- **entry/do/exit behavior indication**: off
- **Simple State**: play
- **States Transition with guards**: stop

Transitions:
- off to on: [change(hmi.off) or (not powered and batteryLevel < U.U1)]
- on to stop: [change(hmi.play)]
- stop to play: [change(hmi.stop)]
ModelicaML: Conditional Eq./Alg. Diagram

Calculate battery status

Start

Explicit state dependency

Conditional flow (if/when) with guards

battery is used (discharging)

battery is not used (charging)

battery status doesn’t change

Set of equations or assignments

(Player Example)
ModelicaML: Requirements Formalization

Two-Tanks System Example

Textual Requirements

- **Max level of liquid in a tank**
  - Text: The level of liquid in a tank shall never exceed 80% of the tank-height.
  - Variables:
    - `maxLevel`: ModelicaReal
    - `tank_height`: ModelicaReal
    - `level`: ModelicaReal

- **Volume of the tank1**
  - Text: The volume of the tank1 shall be 0.8m³.
  - Variables:
    - `tank_volume`: ModelicaReal
    - `design_value`: ModelicaReal
ModelicaML: Requirements Evaluation

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

- tank-height is 0.6m
- Request 001 for tank2 is violated
- Request 001 for tank1 is not violated

Plot Window
ModelicaML

Modeling Example
Example: Two Tanks System

- From the “source” the liquid flows into the “tank1”
- Controller “piContinuous1” controls the level of liquid based on a predefined reference value
- Liquid flows from “tank1” into “tank2”
- Controller “piContinuous2” controls the level of liquid in “tank2”

Required Installations

- OMC from http://www.openmodelica.org/index.php/download
- Eclipse 3.6 (Helios) Modeling from www.eclipse.org
ModelicaML PapyrusMDT: Overview
ModelicaML PapyrusMDT: Overview

1. Model Browser: Shows model elements
2. Properties View: Shows the properties of elements
3. Diagram Editors
4. Palette (different for each diagram type)
5. ModelicaML code generation and validation buttons
6. Component tree: shows the components hierarchy of the selected class
Create ModelicaML Project

- Open Eclipse
- Change the Perspective to Papyrus Perspective
Create ModelicaML Project

Go to File -> Create …
Papyrus Project Files Structure

Diagram file
(can only be edited using Papyrus MDT)

Model file
(Can be edited using any UML2 tool)
Create ModelicaML Project

- Change to Papyrus “Model Explorer”
- Rename the top level element from “model” to (“model” is a Modelica key word!)
- Go to Properties
- Apply ModelicaML profile
Create ModelicaML Project

1. Open ModelicaML example
2. Select ModelicaML profile
3. Apply profiles
4. Choose ModelicaML profile
Configure Model Explorer
Add ModelicaML Specific Views
Create Model Structure

• Create Packages and Classes using ModelicaML menus
Hint: Setting type of components

ModelicaML Primitive Types are:
- ModelicaReal
- ModelicaInteger
- ModelicaString
- ModelicaBoolean
Hint: Setting type of components
Hint: Setting of the component properties: Declaration, Causality, Variability, etc.
Create Class Components

=> Set variability to "parameter"
=> Set causality to "input"
=> Set declaration
Create Function Arguments

1 => Set causality to “input”

2 => Set causality to “output”
Create Class Diagram
Configure Diagram Palette

right click here…
Create Extends Relation

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

BaseController
- components (7)
- behavior (1)
- Equations (Code)

Library
- BaseController
- limitValue

Create Behavior
- Equations

<equationsCode>
error = ref - cIn.val;
withOut.act = outCtrl;

<function>
lim := if p>pMax then pMax
delse if p<pMin then pMin
delse p;

ModelicaML Validation
Create Behavior

- **Components**
  - LiquidSource
    - components (2)
    - behavior (1)
      - eq: set outgoing flow level
  - PContinuousController
    - components (1)
    - behavior (1)
      - eq: calculate

- **Equations**
  - UML
  - ModelicaML
  - Equations
    - eq: set outgoing flow level
    - OutFlow = if time > 150 then 3*flowLevel else flowLevel;
    - der(x) = error/T;
    - outCtrl = K*(error + x);
Create Behavior

### Examples

#### Tank

- **Components:** 10
- **Behavior:** 1
  - **Equation:** balance equation

#### PI Continuous Controller

- **Components:** 1
- **Behavior:** 1
  - **Equation:** calculate

### Equations

**Tank:**

- \( \text{dev}(\text{levelOfLiqui}) = \frac{(\text{qInFlow} - \text{qOutFlow})}{\text{area}} \)  // Mass balance equation

**PI Continuous Controller:**

- \( \text{eq} = \text{calculate} \)

- \( \text{outCtrl} = K_\text{p} (\text{error} + \text{outCtrl}) \)
Create State Machine
Configure Diagram Palette

right click here…
Create State Machine
Create System Architecture

- Create a ModelicaML Connection Diagram under TanksConnectedPI
- 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
Create System Architecture

1. Create components using the palette tools
2. Set component type
3. Drag & drop the port into component
4. Connect components
Model System Architecture
Hint: Element Appearance: Hide the name of the connection stereotype

Select an element
Define Component Modifications

1. Select the component to modify.
2. Modify the properties of the selected component.
3. Open the modification editor.
4. Apply the modifications.
Define Component Modifications

- **Component**: LiquidSource
  - *flowLevel*: 0.02

- **Component**: Tank1
  - *area*: 1

- **Component**: Tank2
  - *area*: 1.3

- **Component**: Controller2
  - *ref*: 0.4
Validate Model
Simulate Model
Simulate Model