Experiments and an FMI Idea on the Online-Minimum Problem

17th Modelisax Meeting

Thomas Beutlich
01/11/2017

get it right®

www.esi-group.com

Problem Description

- Online-computation of the minimum of a (continuous) variable over time in Modelica
- Previous ideas were based
  - State events
  - Sampling
  - Derivatives
- However, no satisfying, obvious or Modelica-builtin solution exists

Long Problem History

- Raised by Mike Tiller at Modelica Trac in Oct. 2008
  https://trac.modelica.org/Modelica/ticket/109
- Rediscovered and presented by Christoph Clauß at Modelisax in May 2014
  http://www.modelisax.de/wp-content/uploads/2014/05/Minimalwertbildung.pdf
- Asked at Stack Overflow in June 2015
  https://stackoverflow.com/q/30732774/8520615
- Asked again by Rene Just Nielsen at Stack Overflow in May 2016
  https://stackoverflow.com/q/37409300/8520615
- Solved by Volker Waurich in March 2017 using the
  https://github.com/vwaurich/ExternalMemoryLib
- Presented by Georg Heß at Modelisax in March 2017
  www.modelisax.de/wp-content/uploads/2017/03/External_Memory.pdf

FMU Based Idea

- Issue with ExternalMemoryLib: Detection of valid solver step
- General: Solver step-size related information (e.g., is a step a valid step or get value at the last valid step) is unspecified in Modelica for good reasons
- But
  - SimulationX has (non-Modelica conform) *last* operator
  - FMI 2.0 offers *fmi2CompletedIntegratorStep* interface function
- Idea: Combine both solutions to create a Modelica conform way based on FMU

- This is where the new trouble started …

Creation of the Last.fmu

- Export model

```
model Last "Last.ism"
  parameter Real y0=0.0 "Signal output initialization";
  SignalBlocks.Function function1(F=if initial() then y0 else last(self.x));
end Last;
```

  Code-Export as FMU 2.0 for Model-Exchange with SimulationX 3.7 and 3.8 failed
- Code-Export with SimulationX 3.6 finally worked out, however
  - Only as FMU 1.0 for Model-Exchange
  - Generated code needed to be manually adapted in order to properly consider the initial value *y0*
  - Manually rename *function1.x* to *u*, *function1.y* to *y* and *y0* to *y_start* in the modelDescription.xml

Import of the Last.fmu

- Issue: Import of an FMU is tool-dependent
- Idea: Import the FMU manually in different tools and create a somehow generic library covering the tool-dependent import models for
  - Dymola
  - OpenModelica
  - SimulationX
- Result: https://github.com/beutlich/LastLib

Import of the Last.fmu in SimulationX

- SimulationX generates the non-Modelica conform keyword *nondiscrete* on Boolean and Integer auxiliary variables, which cause syntax errors in other tools
- Luckily, commenting *nondiscrete* still worked out
- SimulationX always uses single-quoted identifiers
- Simulation requires the original FMU

Import of the Last.fmu in Dymola

- Dymola generates platform-dependent code, i.e., LastLib currently covers the Windows OS
- Dymola requires the unpacked FMU binaries

Import of the Last.fmu in OpenModelica

- OMC generates runtime-dependent code, i.e., LastLib currently covers the C runtime
- OMC requires the unpacked FMU binaries along with the modelDescription.xml

- Credits for Volker Waurich for import, test and verification

Development of the LastLib

- Create a common base class holding the interface variables *u*, *y* and *y_start*

```
partial model LastBase
  parameter Real y_start=0.0 "Signal output initialization";
  Modelica.Blocks.Interfaces.RealInput u "Signal input"
  annotation(Placement(transformation(extent={{-124,-20},{-84,20}})));
  Modelica.Blocks.Interfaces.RealOutput y "Signal output"
  annotation(Placement(transformation(extent={{100,-20},{140,20}})));
end LastBase;
```

- Manually extend tool-dependent import models from *LastBase*
- Create a Resources directory
- Fix all FMU/binaries pathes using *Modelica.Utilities.Files.loadResource*

Development of the Minimum block

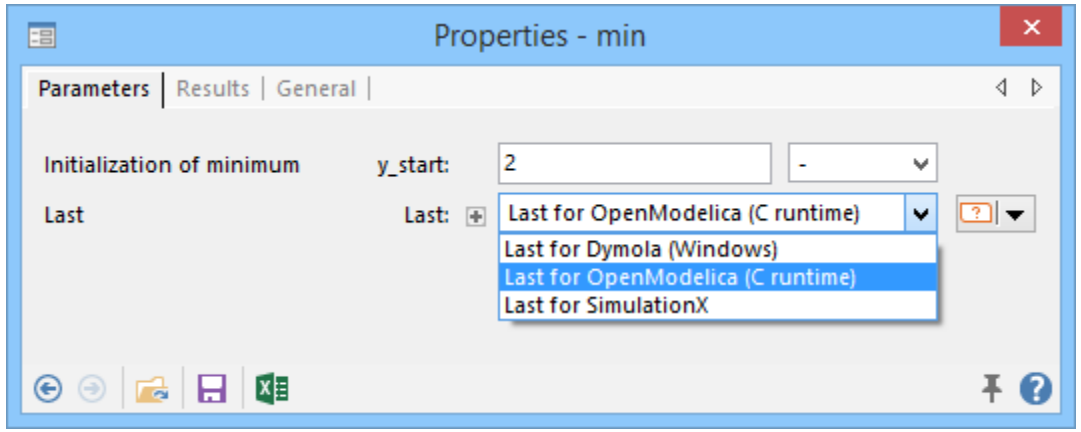
- Create a minimum block comprising all of the tool-dependent import models

```
block Minimum
  extends Modelica.Blocks.Interfaces.SISO;
  parameter Real y_start=Modelica.Constants.inf "Initialization of minimum";
  replaceable model Last=LastLib.Last_omc constrainedby LastLib.LastBase(y_start=y_start) "Last FMU"
  annotation(choices(
    choice(redeclare model Last=LastLib.Last_dymola "Last for Dymola (Windows)"),
    choice(redeclare model Last=LastLib.Last_omc "Last for OpenModelica (C runtime)"),
    choice(redeclare model Last=LastLib.Last_simx "Last for SimulationX")));
  Last last "Last FMU"
  annotation(Placement(transformation(extent={{-30,10},{-10,30}})));
  Modelica.Blocks.Math.Min min annotation(Placement(transformation(extent={{10,-10},{30,10}})));
  equation
    connect(u, min.u2);
    connect(min.y, y);
    connect(last.y, min.u1);
    connect(min.y, last.u);
  annotation(defaultComponentName="min", preferredView="diagram");
end Minimum;
```

Verification of the Minimum block

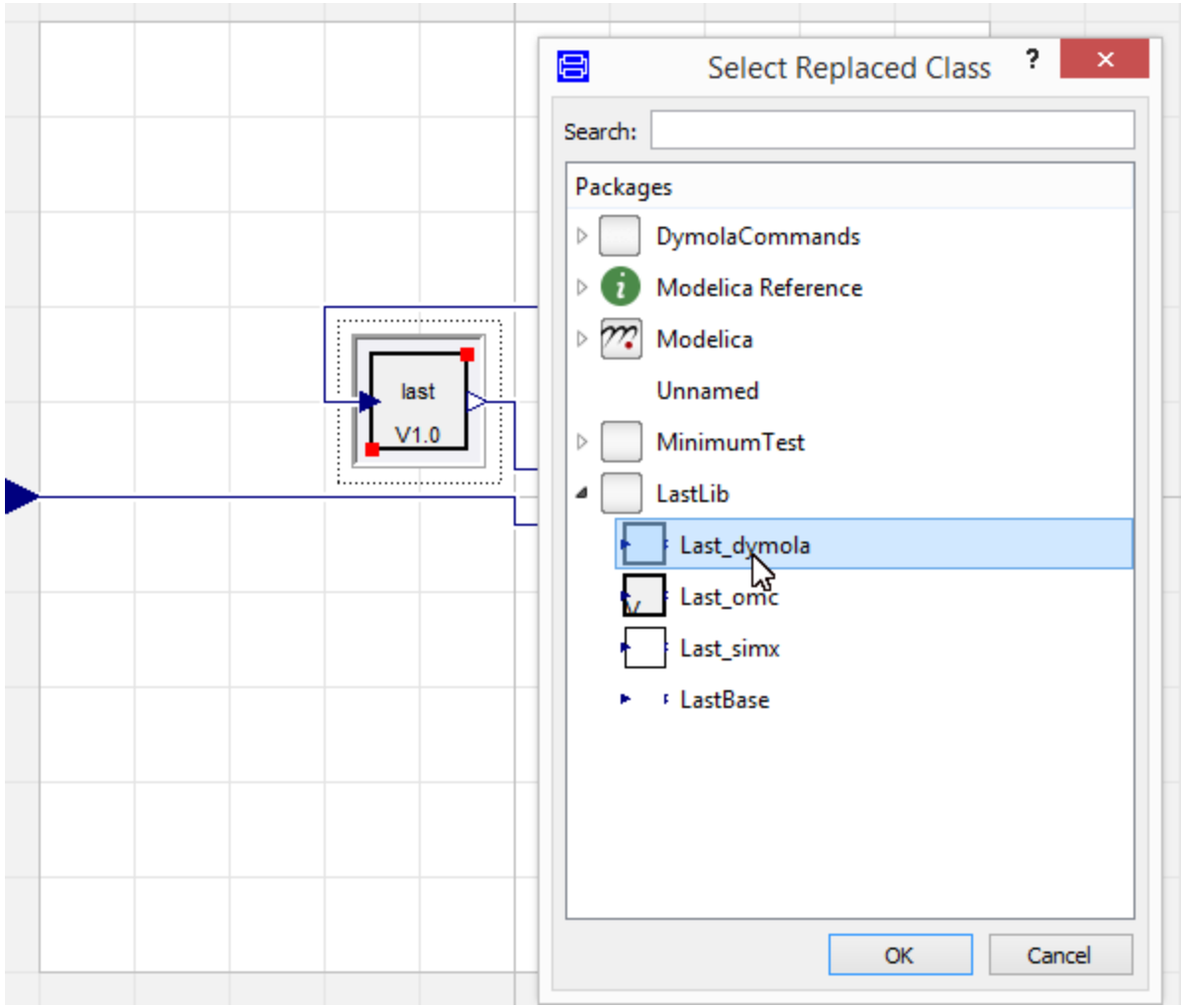
- Verify using the test model from Christoph Clauß

```
model Test
  Real x(start=3, fixed=true);
  Real v(start=-1, fixed=true);
  Modelica.Blocks.Sources.RealExpression realExpression(y=x)
  annotation(Placement(transformation(extent={{-30,40},{-10,60}})));
  Minimum min(y_start=2)
  annotation(Placement(transformation(extent={{10,40},{30,60}})));
  equation
    connect(realExpression.y, min.u);
  equation
    der(x) = v;
    der(v) = -x + 1 - time/10;
  annotation(experiment(StopTime=20));
end Test;
```



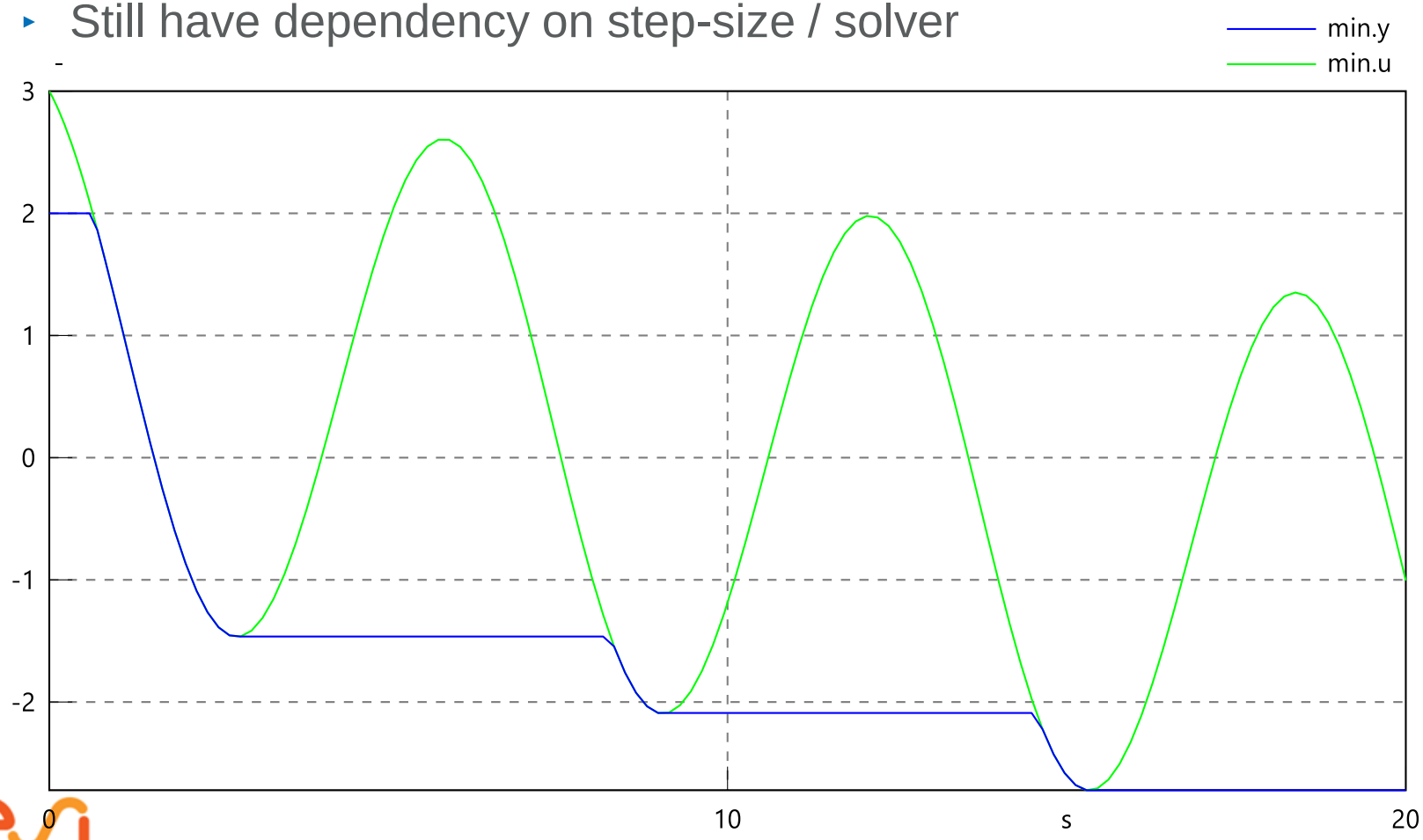
Verification of the Minimum block (continued)

- Crashes in Dymola
  - Workaround: Ignore the redeclared model Last, but directly apply „Change Class“ of from *Last* to *LastLib.Last_dymola* and manually set the *y_start* modification in block *Minimum*
    ```
    LastLib.Last_dymola
    last(y_start=y_start);
    ```
- Runs in OMC (but initialization is wrong since the modification of the constrainedby is ignored)
  - Woraround: Set the *y_start* modification again at the instance in block *Minimum*
    ```
    Last last(y_start=y_start);
    ```
- Runs successfully in SimulationX



Verification of the Minimum block (continued)

- As expected
  - No event iteration
  - Still have dependency on step-size / solver



Summary

- Modelica is open standard for modeling and simulation of cyber-physical systems and supported by several tools
- FMI is open standard for model distribution and supported by dozens of tools
- But, sharing Modelica models with imported FMUs between Modelica tools is pain
- MA needs to address https://trac.modelica.org/Modelica/ticket/1626 (by https://trac.modelica.org/Modelica/ticket/1727) to handle imported FMUs in a portable way

FMU Import Proposal

- Have a general Modelica block for FMU import with standardized interface

```
model FMUImport
  parameter String fmuPath="somewhere/Last.fmu";
  parameter String fmuInstanceName="last_fmu";
  parameter ModelicaServices.FMISettings fmuSettings(logging=true);
  parameter Real pr[ModelicaServices.getRealParams(fmuPath)] "Real params";
  parameter Integer pi[ModelicaServices.getIntegerParams(fmuPath)] "I. p.";
  input Real ur[ModelicaServices.getRealInputs(fmuPath)] "Real inputs";
  input Integer ui[ModelicaServices.getIntegerInputs(fmuPath)] "I. inputs";
  output Real yr[ModelicaServices.getRealOutputs(fmuPath)] "Real outputs";
  output Integer yi[ModelicaServices.getIntegerOutputs(fmuPath)] "I. o.";
  equation
  // ...
end FMUImport;
```

- Implementation goes to in ModelicaServices, i.e., needs to be handled by tool vendors (in a tool-specific way)