New Features in Modelica 3.4

15th Modelisax Meeting

Thomas Beutlich
03/01/2017

get it right®

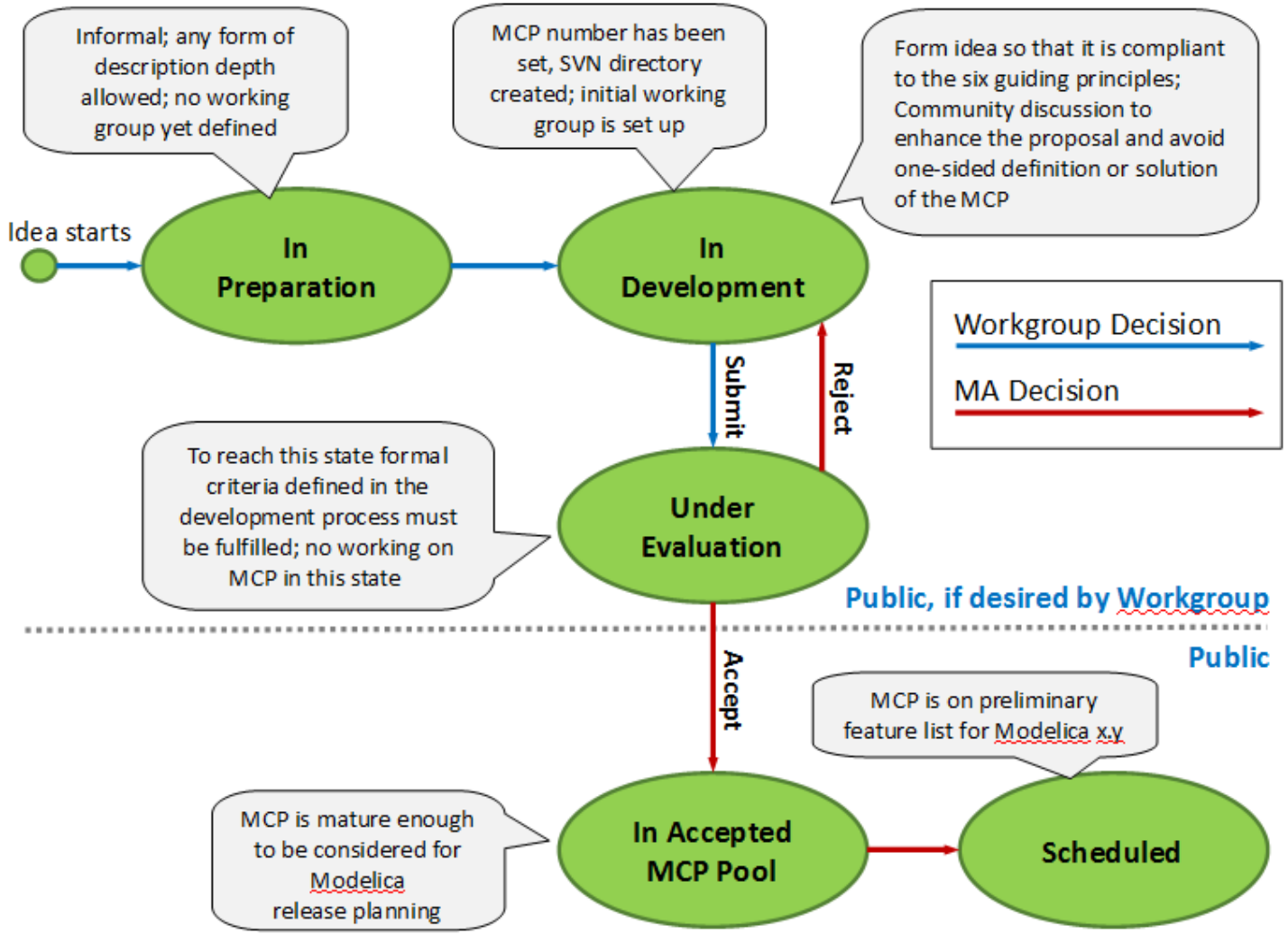
www.esi-group.com

Introduction / Roadmap

- Modelica 3.4 will be the first Modelica language specification that considered the Modelica Change Proposal (MCP) process (for 7 new features)
- More than 150 issues closed (mainly clarifications / corrections and minor additions): https://trac.modelica.org/Modelica/milestone/ModelicaSpec3.4
- Coordinated by Hans Olsson, project leader of the Modelica Association Language Project

- Preliminary feature list on February 5, 2017
- Beta release on February 20, 2017: https://svn.modelica.org/projects/ModelicaDesign/trunk/ModelicaTask-LangSpec/ModelicaSpec/_ModelicaSpec34.pdf
- Release candidate on March 20, 2017
- Modelica 3.4 release on April 10, 2017
- Modelica Conference: May 15 - 17, 2017

MCP Process



Automatic Conversions between Different Versions (MCP-0014)

- Section 18.8.2
- Standardize conversion annotation and conversion rules/script for automatic library upgrading as already available in MSL and supported by Dymola
- Rules
  - convertClassIf("OldClass", "oldElement", "whenValue", "NewClass")
  - convertClass("OldClass","NewClass")
  - convertElement("OldClass","OldName","NewName")
  - convertModifiers("OldClass",{"OldModifier1=default1","OldModifier2=default2",...}, {"NewModifier1=...%OldModifier1%"})
  - convertMessage("OldClass", "Failed Message");

```
package Modelica
  ...
  annotation(version="3.1",
             conversion(noneFromVersion="3.1  Beta 1",
                        noneFromVersion="3.1  Beta 2",
                        from(version={"2.1", "2.2", "2.2.1"},
                             script="convertTo3.mos"),
                        from(version="1.5",
                             script="convertFromModelical_5.mos")));
end Modelica;
```

Flattening is Clearly Specified (MCP-0019)

- Sections 5.3.1 and 5.6
- Describes the flattening process for the first time
- Mainly relevant for tool vendors
- The flattening proceeds in two major steps
  1. Instantiation process
     1. Class Tree
     2. Instance Tree
     3. Instantiation Procedure
     4. Steps of Instantiation
  2. Generation of the flat equation system

Convert from Integer to Enumeration (MCP-0022)

- Primarily section 4.8.5.3
- Added missing type conversion from Integer to enumeration value (since Modelica < 3.4 only knew about type conversion from enumeration value to Integer or String)

```
type Colors = enumeration ( RED, GREEN, BLUE, CYAN, MAGENTA, YELLOW );

i = Integer(Colors.GREEN);
c1 = Colors(i);
c2 = Colors(10); // An error
```

Explicit Cast of a Model to Record (MCP-0023)

- Section 12.6.1
- Outcome of the MODRIO research project for requirements modeling
- Convenience feature (already was possible in Modelica < 3.4)

```
connector Flange
   Real phi;
   flow Real tau;
end Flange;

model Submodel1
   Real m1;
   Boolean b1;
   Flange flange;
end Submodel1;

model Model2
  Real    r1;
  Real    r2;
  Integer i2
  Pin p1, p2;
  Submodel1 sub1;
protected
  Integer i1;
  ...
end Model2;
```

```
record MyFlange
   Real tau;
end MyFlange;

record MySubRecord1
   Boolean  b1;
   MyFlange flange;
end MySubRecord1;

record MyRecord2
  Real    r1;
  Integer i2;
  MySubRecord1 sub1;
end MyRecord2;
```

```
model Model
  Model2 s1;
  Model2 s2[2];
  MyRecord2 rec1    = MyRecord2(s1);
  MyRecord2 rec2[2] = MyRecord2(s2);
  …
end Model;

// Model is conceptually mapped to
model ModelExpanded
   Model2 s1;
   Model2 s2[3];
   MyRecord2 rec1    = MyRecord2(r1=s1.r1, i2=s1.i2,
                                 sub1 = MySubRecord1(b1=s1.sub1.b1,
                                   flange = MyFlange(tau=s1.sub1.flange.tau));
   MyRecord2 rec2[2] = {MyRecord2(r1=s2[1].r1, i2=s2[1].i2,
                                 sub1 = MySubRecord1(b1=s2[1].sub1.b1,
                                    flange = MyFlange(tau=s1[1].sub1.flange.tau)),
                       MyRecord2(r1=s2[2].r1, i2=s2[2].i2,
                                 sub1 = MySubRecord1(b1=s2[2].sub1.b1,
                                    flange = MyFlange(tau=s2[2].sub1.flange.tau)};
   ...
end ModelExpanded;
```

Initialization of Clocked Continuous States (MCP-0024)

- Section 16.8.1 and 16.10
- Clarified, simplified and unified the treatment of start values of discrete states in clocked model partitions
- The new firstTick operator returns true at the first tick of the clock of the expression, in which this operator is called, otherwise false.

A continuous-time model is inverted, discretized and used as feedforward controller for a PI controller (der(..), previous, interval are used in the same partition):

```
block MixedController
   parameter Real T "Time constant of continuous PI controller";
   parameter Real k "Gain of continuous PI controller";
   input  Real y_ref, y_meas;
   Real y;
   output Real yc;
   Real z(start=0);
   Real xc(start=1, fixed=true);
   Clock c = Clock(Clock(0.1), solverMethod="ImplicitEuler");
protected
   Real uc;
   Real Ts = interval(uc);
equation
   /* Continuous-time, inverse model */
   uc = sample(y_ref, c);
   der(xc) = uc;

   /* PI controller */
   z = if firstTick() then 0 else previous(z) + Ts/T*(uc - y_meas);
   y = xc + k*(xc + uc);
   yc = hold(y);
end MixedController;
```

Options to Ellipse Annotation (MCP-0026)

- Section 18.6.5.4

```
type EllipseClosure = enumeration(None, Chord, Radial);

 record Ellipse
   extends GraphicItem;
   extends FilledShape;
   Extent extent;
   Real startAngle(quantity="angle", unit="deg")=0;
   Real endAngle(quantity="angle", unit="deg")=360;
   EllipseClosure closure = if startAngle == 0 and endAngle == 360
                            then EllipseClosure.Chord
                            else EllipseClosure.Radial;
 end Ellipse;
```

Added Specific Font Names for Text Annotation

- Section 18.6.5.5
- The font names "serif", "sans-serif", and "monospace" shall be recognized.

```
record Text
  extends GraphicItem;
  extends FilledShape;
  Extent extent;
  String textString;
  Real fontSize = 0 "unit pt";
  String fontName;
  TextStyle textStyle[:];
  Color textColor=lineColor;
  TextAlignment horizontalAlignment = TextAlignment.Center;
end Text;
```

Allow Mixed Real and non-Real Record Derivatives (MCP-0028)

- Section 12.7.1
- Relevant in ThermoDynamic libraries
- Legalize and standardize Record use in derivative functions with mixed Real and non-Real (= Integer, Boolean, Enumeration, String or Record) type components passed as inputs or outputs
- Records for the derivative function do only care for the Real components

```
record ThermodynamicState "Thermodynamic state"
  SpecificEnthalpy h "Specific enthalpy";
  AbsolutePressure p "Pressure";
  Integer phase(min=1, max=2, start=1);
end ThermodynamicState;

record ThermoDynamicState_der "Derivative"
  Real h "Specific enthalphy derivative";
  Real p "Pressure derivative";
  // Integer input is skipped
end ThermodynamicState_der;
```

```
function density
  input ThermodynamicState state "Thermodynamic state";
  output Density d "Density";
algorithm
  ...
  annotation(derivative=density_der);
end density;

function density_der
  input ThermodynamicState state "Thermodynamic state";
  input ThermodynamicState_der state_der;
  output DensityDerivative d "Density derivative";
algorithm
  ...
end density_der;
```

Added unbound Attribute for Real

- Section 4.8.1
- Normal error tolerance considers absolute and relative error:
  - <= tol*(|nominal(x)|+|x|)
- If unbounded only consider absolute tolerance (and skip relative error):
  - <= tol*|nominal(x)|

```
type Real // Note: Defined with Modelica syntax although predefined
  RealType value;                        // Accessed without dot-notation
  parameter StringType  quantity    = "";
  parameter StringType  unit        = ""  "Unit used in equations";
  parameter StringType  displayUnit = ""  "Default display unit";
  parameter RealType    min=-Inf, max=+Inf;  // Inf denotes a large value
  parameter RealType    start = 0;           // Initial value
  parameter BooleanType fixed = true,  // default for parameter/constant;
                        = false; // default for other variables
  parameter RealType    nominal;             // Nominal value
  parameter BooleanType unbounded=false; // For error control
  parameter StateSelect stateSelect = StateSelect.default;
equation
  assert(value >= min and value <= max, "Variable value out of limit");
end Real;
```

Added fixed Attribute for String

- Section 4.8.4
- Add missing fixed attribute for String type (similarly as for Integer or Boolean)

```
type String // Note: Defined with Modelica syntax although predefined
  StringType value;                    // Accessed without dot-notation
  parameter StringType  quantity    = "";
  parameter StringType start = "";     // Initial value
  parameter BooleanType fixed = true,  // default for parameter/constant;
                            = false, // default for other variables
end String;
```

```
model Model
  discrete output String data(start="", fixed=true);
  equation
    when sample(0, 0.1) then
      data = String(time);
    end when;
end Model;
```

Allow Multiple Include and Library Directories

- Section 12.9.4
- IncludeDirectory and LibraryDirectory attribute of the external function annotation can be vectors (previously: only scalar)

```
function readVarFromMatFile
  input String fileName;
  input String varName;
  output Real var;
  external "C" var = getVar(fileName, varName) annotation(
    Library = {"mat", "hdf5", "zlib"},
    IncludeDirectory = {"/usr/include/mat", "/usr/include/hdf5", "/usr/include/zlib"});
end readVarFromMatFile;
```

Allow Specific Libraries for Different Compilers or Versions

- Section 12.9.4
- Previously: In Modelica < 3.4 the compiler/linker started by the Modelica tool searches external libraries in the path given by the LibraryDirectory attribute or platform-specific subdirectories (win32/win64/linux32/linux64).
- The "win32"/"win64" directories may contain "gcc47", "vs2010", "vs2012" for specific versions of these compilers and these are used instead of the general "win32"/"win64" directories, and similarly for other platforms.

Enable Specific C Standards

- Sections 12.9 and 12.9.6
- Not yet finalized
- Previously: The language specification of the (external) function must currently be one of "builtin", "C" or "FORTRAN 77“ (where "C" means "C89")
- Explicitly allow "C89" and enable "C99" and "C11" for newer C standards.

Added External Warning Functions

- Section 12.9.6
- Previously: ModelicaMessage and ModelicaError (and ist format variants, e.g., ModelicaFormatMessage)
- Added ModelicaWarning (and format variants) (and treat it similarly as an assert with level=AssertionLevel.Warning in the Modelica code)

Summary

- About 3 years after Modelica 3.3 rev. 1 a new language standard will be released (within second quarter of 2017) containing hundreds of
  - Clarifications
  - Corrections
- and
  - Few minor or simply convenience features
  - One major new feature: Standardization of the conversion script

- It turned out that the MCP process itself needs improvement. Also, some features were added by MCP, other features went directly in Modelica 3.4 w/o MCP.

- In the Modelica Association Library Project it was discussed at the 93rd Modelica Design Meeting (February 27, 2017) that the next Modelica Standard Library will be based on Modelica 3.4 – with explicit exclusion of the clocked/synchronous and state machine features of Modelica 3.3.

Thanks for contributing!