Mattsson S.E., Elmqvist H., Otter M., and Olsson H.:
Initialization of Hybrid Differential-Algebraic Equations in Modelica 2.0
2nd International Modelica Conference, Proceedings, pp. 9-15

Paper presented at the 2nd International Modelica Conference, March 18-19, 2002,
Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Oberpfaffenhofen, Germany.

All papers of this workshop can be downloaded from
http://www.Modelica.org/Conference2002/papers.shtml

Initialization of Hybrid Differential-Algebraic Equations in Modelica 2

Sven Erik Mattsson[†], Hilding Elmqvist[†], Martin Otter[‡] and Hans Olsson[†]
[†]Dynasim AB, Lund, Sweden, {svenerik, elmqvist, hans}@dynasim.se
[‡]DLR, Oberpfaffenhofen, Germany, martin.otter@dlr.de

Abstract

Modelica 2 provides new powerful language constructs for specifying initial conditions. Before any operation is carried out with a Modelica model, such as simulation or linearization, initialization takes place to assign consistent values for all variables, derivatives and pre-variables present in the model. To obtain consistent values, the initialization uses all equations and algorithms that are utilised during the simulation. Additional constraints necessary to determine the initial values of all variables can be provided as start values for any variables as well as additional constraint equations in initial equation sections. A novel feature is the possibility to have a sampled controller initialized in steady state. This tutorial paper describes and explains the new language constructs and illustrates how they in combination with Modelica's other language elements allow very flexible and powerful initialization conditions to be defined.

1. Introduction

A dynamic model describes how the states evolve with time. The states are the memory of the model, for example in mechanical systems positions and velocities. When starting a simulation, the states need to be initialized.

For an ordinary differential equation, ODE, in state space form, d**x**/dt = **f**(**x**, t), the state variables, **x**, are free to be given initial values. However, more flexibility in specifying initial conditions than setting state variables is needed. In many cases we would like to start at steady state implying that the user specifies d**x**/dt = 0 as initial condition to get the initial values of **x** calculated automatically by solving **f**(**x**, t) = 0. Besides the states, a model has also other variables and in many cases it is natural to specify initial conditions in terms of these variables.

In January 2002, Modelica 2 was released [3]. The new language constructs permit flexible specification of initial conditions as well as the correct solution of difficult, non-standard initialization problems occurring in industrial applications. Modelica 2 provides a mathematically rigid specification of the initialization of hybrid differential algebraic equations.

Dymola [1,2] supports the new language constructs of Modelica 2. Earlier Dymola versions had pure numeric support for initialization. Experiences from industrial applications including closed kinematics loops and thermodynamic problems showed that this was not sufficient. The numerical solvers were often not able to solve the large and non-linear problems. Now Dymola also manipulates symbolically the initialization problem and generates analytic Jacobians for nonlinear subproblems. Experience indicates that this approach is more robust and reliable. Moreover, the special analysis of the initialization problem allows Dymola to give diagnosis and user guidance when the initialization problem turns out not to be well posed.

This paper describes the language constructs to specify initial conditions and examples for the usage are given.

2. Basics

Before any operation is carried out with a Modelica model, especially simulation, initialization takes place to assign consistent values for all variables present in the model. During this phase, also the derivatives, **der**(…), and the **pre**-variables, **pre**(…), are interpreted as unknown algebraic variables. The initialization uses all equations and algorithms that are utilized during the simulation.

Additional constraints necessary to determine the initial values of all variables can be provided in two ways:

1. Start values for variables
2. Initial equations and initial algorithms

For clarity, we will first focus on the initialization of continuous time problems because there are some differences in the interpretation of the start values of continuous time variables and discrete variables. Also there are special rules for the usage of when clauses during initialization. All this makes it simpler to start discussing pure continuous time problems and after that discuss discrete and hybrid problems.

3. Continuous time problems

Initial equations and algorithms

Variables being subtypes of Real have an attribute *start* allowing specification of a start value for the variable

```
Real v(start = 2.0);
parameter Real x0 = 0.5;
Real x(start = x0);
```

The value for start shall be a parameter expression.

There is also another Boolean attribute *fixed* to indicate whether the value of start is a *guess* value (fixed = **false**) to be used in possible iterations to solve nonlinear algebraic loops or whether the variable is required to have this value at start (fixed = **true**). For constants and parameters, the attribute fixed is by default **true**, otherwise **fixed** is by default false.

For a continuous time variable, the construct

```
Real x(start = x0, fixed = true);
```

implies the additional initialization equation

```
x = x0;
```

Thus, the problem

```
  parameter Real a = -1, b = 1;
  parameter Real x0 = 0.5;
  Real x(start = x0, fixed = true);
equation
  der(x) = a*x + b;
```

has the following solution at initialization

```
a     := -1;
b     := 1;
x0    := 0.5;
x     := x0;       // = 0.5
der(x):= a*x + b;  // = 0.5
```

Initial equations and algorithms

A model may have the new sections **initial equation** and **initial algorithm** with additional equations and assignments that are used solely in the initialization phase. The equations and assignments in these initial sections are viewed as pure algebraic constraints between the initial values of variables and possibly their derivatives. It is not allowed to use when clauses in the initial sections.

Steady state

To specify that a variable x shall start in steady state, we can write

```
initial equation
  der(x) = 0;
```

A more advanced example is

```
  parameter Real x0;
  parameter Boolean steadyState;
  parameter Boolean fixed;
  Real x;
initial equation
  if steadyState then
    der(x) = 0;
  else if fixed then
    x = x0;
  end if;
```

If the parameter steadyState is **true**, then x will be initialized at steady state, because the model specifies the initialization equation

```
initial equation
  der(x) = 0;
```

If the parameter steadyState is **false**, but fixed is **true** then there is an initialization equation

```
initial equation
  x = x0;
```

If both steadyState and fixed are **false**, then there is no initial equation.

The approach as outlined above, allows x0 to be any expression. When x0 is a parameter expression, the specification above can also be given shorter as

```
  parameter Real x0;
  parameter Boolean fixed;
  parameter Boolean steadyState;
  Real x(start = x0,
         fixed = fixed and
                 not steadyState);
initial equation
  if steadyState then
    der(x) = 0;
  end if;
```

Mixed Conditions

Due to the flexibility in defining initialization equations in Modelica 2, it is possible to formulate more general initial conditions: For example, an aircraft needs a certain minimum velocity in order that it can fly. Since this velocity is a state, a useful initialization scheme is to provide an initial velocity, i. e., an initial value for a *state*, and to set all other *state derivatives* to zero. This means, that a mixture of initial states and initial state derivatives is defined.

How many initial conditions?

How many initial conditions are needed for a continuous time problem?

For an ordinary differential equation, ODE, in state space form, d**x**/dt = **f**(**x**, t), *exactly* dim(**x**) additional conditions are needed, in order to arrive at 2*dim(**x**) equations for the 2*dim(**x**) unknowns **x**(t_0) and d**x**/dt(t_0).

The situation is more complex for a system of differential algebraic equations, DAE,

$$\mathbf{0} = \mathbf{g}(d\mathbf{x}/dt, \mathbf{x}, \mathbf{y}, t)$$

where **x**(t) are variables appearing differentiated, **y**(t) are algebraic variables and dim(**g**) = dim(**x**) + dim(**y**). Here it can only be stated that *at most* dim(**x**) additional conditions **h**(..) are needed in order to arrive at 2*dim(**x**)+dim(**y**) equations for the same number of unknowns, d**x**/dt(t_0), **x**(t_0), **y**(t_0):

$$\mathbf{0} = \begin{bmatrix} \mathbf{g}(\dot{\mathbf{x}}(t_0), \mathbf{x}(t_0), \mathbf{y}(t_0), \mathbf{t}_0) \\ \mathbf{h}(\dot{\mathbf{x}}(t_0), \mathbf{x}(t_0), \mathbf{y}(t_0), \mathbf{t}_0) \end{bmatrix}$$

The reason is that the DAE problem may be a higher index DAE problem, implying that the number of continuous time states is less than dim(**x**).

It may be difficult for a user of a large model to figure out how many initial conditions have to be added, especially if the system has higher index. At translation Dymola performs an index reduction and selects state variables. Thus, Dymola establishes how many states there are. If there are too many initial conditions, Dymola outputs an error message indicating a set of initial equations or fixed start values from which initial equations must be removed or start values inactivated by setting fixed = **false**.

If initial conditions are missing, Dymola makes automatic default selection of initial conditions. The approach is to select continuous time states with inactive start values and make their start values active by turning their fixed attribute to **true** to get a structurally well posed initialization problem. A message informing about the result of such a selection can be obtained.

Interactive setting of start values

The initial value dialogue of the Dymola main window has been redesigned. Previously, it included all continuous time states. Now it includes the continuous time variables having active start values i.e., fixed=**true** and the start value being a literal. Setting parameters may of course influence an active start value bound to a parameter expression.

When setting variables from scripts Dymola generates a warning if setting the variable has no effect what so ever, e.g. if it is a structural parameter.

Non-linear algebraic loops

A non-linear algebraic problem may have several solutions. During simulation a numerical DAE solver tends to give the smoothest solution. A DAE solver is assumed to start at a consistent point and its task is to calculate a new point along the trajectory. By taking a sufficiently small step and assuming the existence of a Jacobian that is non-singular there is a local well-defined solution.

The initialization task is much harder and precautions must be taken to assure that the correct solution is obtained. The means to guide the solver include min and max values as well as start values for the unknowns.

As a simple example, consider a planar pendulum with fixed length L.

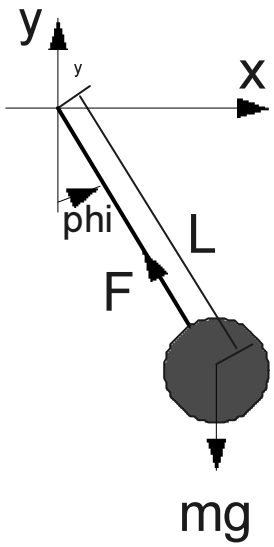

Figure 1: A planar pendulum.

The position of the pendulum can be given in polar coordinates. Introduce an angle, phi, that is zero, when the pendulum is hanging downward in its rest position. The model can be given as

```
  parameter Real g = 9.81;
  parameter Real m = 1;
  parameter Real L = 1;
  Real phi, w;
equation
  der(phi) = w;
  m*der(w) = -(m*g/L)*sin(phi);
```

Assume now that we want to specify the initial condition in Cartesian coordinates defined as

```
x =  L*sin(phi);
y = -L*cos(phi);
```

If we define

```
Real y(start = 0; fixed = true);
```

the pendulum will start in a horizontal position. However, there are two horizontal positions, namely

```
x = -L and x = L
```

To indicate preference for a positive value for x, we can define

```
Real x(start = L);
```

It means that we provide a guess value for numerical solvers to start from. They will hopefully find the positive solution for x, because, it is closer to L than the negative solution.

For the angle phi there are many values giving the desired position, because adding or subtracting 2π gives the same Cartesian position. Also, here the start value can be used to indicate the desired solution. How, critical it is to get a special solution depends of course on what phi will be used for in the model and the aim of the simulation. If no start value is given zero is used.

4. Parameter values

Parameters are typically given values in a model through definition equation or set interactively before a simulation. Modelica 2 also allows parameter values to be given implicitly in terms of the initial values of all variables.

Recall the planar pendulum and assume that we would like to specify the initial position as

```
Real x(start = 0.3; fixed = true);
Real y(start = 0.4; fixed = true);
```

This means that we in fact also specify the length of the pendulum to be 0.5. To specify that the parameter L shall be calculated from the initial conditions, we define it as

```
parameter Real L(fixed = false);
```

Recall that the attribute fixed is by default **true** for constants and parameters, otherwise fixed is by default **false**.

The semantics of parameters in Modelica is a variable that is constant during simulation. The possibility to let the parameter value to depend on the initial values of time dependent (continuous-time or discrete) variables does not violate this semantics.

This feature has many useful applications. It allows powerful reparametrizations of models. As an example, consider the model of an ideal resistor. It has one parameter, R, being the resistance. Assume that we would like to have use it as a resistive load with a given power dissipation at a steady state operating point. It is just to extend from the resistor model given in the Modelica Standard Library and

1. Add a parameter P0 to specify the power dissipation.
2. Set fixed=**false** for parameter R.
3. Add an initial equation section with v*i = P0.

In power systems, it is common practice to specify initial conditions in steady state and use different kind of load models including resistive load and specify their steady state operating conditions in terms of active and reactive power dissipation.

In some cases parameters may be provided outside of a Modelica model and the actual values may be read from file or parameter values may be inquired from a database system during initialization:

```
  parameter Real A(fixed=false);
  parameter Real w(fixed=false);
  Real x;
initial equation
  (A,w) = readSineData("init.txt");
equation
   der(x) = -A*sin(w*x);
```

5. Discrete and hybrid problems

The language constructs for specifying initial conditions for discrete variables are as for the continuous time variables: start values and initial equations and algorithms.

Variables being subtypes of Real, Integer, Boolean and String have an attribute start allowing specification of a start value for the variable.

For discrete variables declarations

```
Boolean b(start = false, fixed = true);
Integer i(start = 1,     fixed = true);
```

imply the additional initialization equations

```
pre(b) = false;
pre(i) = 1;
```

This means that a discrete variable v itself does not get an initial value (= $v(t_0+\varepsilon)$), but the **pre**-value of v (= $v(t_0-\varepsilon)$) does.

When clauses at initialization

For the initialization problem there are special semantic rules for **when** clauses appearing in the model. During simulation a **when** clause is only active when its condition becomes **true**. During initialization the equations of a **when** clause are only active during initialization, if the **initial**() operator explicitly enables it.

```
when {initial(), condition1, …} then
   v = …
end when;
```

Otherwise a **when** clause is in the initialization problem replaced by v = **pre**(v) for all its left hand side variables, because this is also the used equation during simulation, when the **when**-clause is not active.

Non-unique initialization

In certain situations an initialization problem may have an infinite number of solutions, even if the number of equations and unknown variables are the same during initialization. Examples are controlled systems with friction, or systems with backlash or dead-zones. Assume for example backlash is present. Then, all valid positions in this element are solutions of steady state initialization, although this position should be computed from initialization. It seems best to not rely on some heuristics of the initializer to pick one of the infinite number of solutions. Instead, the continuous time equations may be modified during initialization in order to arrive at a unique solution. Example:

```
y = if initial() then
       // smooth characteristics
    else
       // standard characteristics
```

Well-posed initialization

At translation Dymola analyses the initialization problem to check if it is well posed by splitting the problem into four equation types with respect to the basic scalar types Real, Integer, Boolean and String and decides whether each of them are well-posed.

As described for the pure continuous-time problem, Dymola outputs error diagnosis in case of over specified problems. In case of under specified problems Dymola makes automatic default selection of initial conditions.

How many initial conditions?

Basically, this is very simple: Every discrete variable v needs an initial condition, because $v(t_0-\varepsilon)$ is otherwise not defined. Example:

```
  parameter Real t1 = 1;
  discrete Real u(start=0, fixed=true);
  Real x(start=0, fixed=true);
equation
  when time > t1 then
     u = ...
  end when;
  der(x) = -x + u;
```

During initialization and before the **when**-clause becomes active the first time, u has not yet been assigned a value by the **when**-clause although it is used in the continuous part of the model. Therefore, it would be an error, if **pre**(u) would not have been defined via the start value in the u declaration.

On the other hand, if u is used solely inside this **when**-clause and **pre**(u) is not utilized in the model, an initial value for u may be provided but *does not influence* the simulation, because the first access of u computes u in the **when**-clause and afterwards u is utilized in other equations inside the **when**-clause, i. e., the initial value is never used.

Since it may be tedious for a modeller to provide initial values for all discrete variables, Modelica 2 only requires to specify initial values of discrete variables which influence the simulation result. Otherwise, a default value may be used.

6. Example: Initialization of discrete controllers

Below four variants to inialize a simple plant controlled by a discrete PI controller are discussed.

Variant 1: Initial values are given explicitly

```
  parameter Real k=10, T=1;
    // PI controller parameters.
  parameter Real Ts = 0.01 "Sample time";
  input     Real xref  "reference input";
            Real x (fixed=true, start=2);
  discrete  Real xd(fixed=true, start=0);
  discrete  Real u (fixed=true, start=0);
equation
  // Plant model
  der(x) = -x + u;

  // Discrete PI controller
  when sample(0, Ts) then
    xd = pre(xd) + Ts/T*(xref - x);
     u = k*(xd + xref - x);
  end when;
```

The model specifies all the initial values for the states explicitly. The when clause is not enabled at initialization but it is replaced by

```
xd      := pre(xd)
u       := pre(u)
```

The initialization problem is thus

```
x       := x.start  // = 2
pre(xd) := xd.start // = 0
pre(u)  := u.start  // = 0
xd      := pre(xd)  // = 0
u       := pre(u)   // = 0
der(x)  := -x + u   // = -2
```

Variant 2: Initial values are given explicitly and the controller equations are used during initialization. It is as Variant 1, but the **when** clause is enabled

```
 // Same declaration as variant 1
equation
 der(x) = -x + u;

 when {initial(), sample(0,Ts)} then
   xd = pre(xd) + Ts/T*(xref - x);
   u  = k*(xd + xref - x);
 end when;
```

It means that the **when** clause appears as

```
xd = pre(xd) + Ts/T*(xref - x);
u  = k*(xd + xref - x);
```

in the initialization problem, which becomes

```
x      := x.start  // = 2
pre(xd):= xd.start // = 0
pre(u) := u.start  // = 0
xd     := pre(xd) + Ts/T*(xref - x);
u      := k*(xd + xref - x);
der(x) := -x + u;
```

Variant 3: As Variant 2 but initial conditions defined by initial equations

```
 discrete  Real xd;
 discrete  Real u;
// Remaining declarations as in variant 1
equation
 der(x) = -x + u;
 when {initial(), sample(0, TS)} then
   xd = pre(xd) + Ts/T*(xref - x);
    u  = k*(xd + xref - x);
 end when;
initial equation
 pre(xd) = 0;
 pre(u)  = 0;
```

leads to the following equations during initialization

```
x      := x.start  // = 2
pre(xd):= 0
pre(u) := 0
xd     := pre(xd) + Ts/T*(xref - x)
u      := k*(xd + xref - x)
der(x)  := -x + u;
```

Variant 4: Steady state initialization

Assume that the system is to start in steady state. For continuous time state, x, it means that its derivative shall be zero; **der**(x) = 0; While for the discrete state, xd, it means **pre**(xd) = xd; and the when clause shall be active during initialization

```
 Real           x (start=2);
 discrete  Real xd;
 discrete  Real u;
// Remaining declarations as in Variant 1
equation
 // Plant model
 der(x) = -x + u;

 // Discrete PID controller
 when {initial(), sample(0, Ts)} then
   xd = pre(xd) + Ts/T*(x - xref);
    u = k*(xd + x - xref);
 end when;
```

```
initial equation
 der(x)  = 0;
 pre(xd) = xd;
```

The initialization problem becomes

```
der(x) := 0

// Linear system of equations in the
// unknowns: xd, pre(xd), u, x
pre(xd) = xd
xd      = pre(xd) + Ts/T*(x - xref)
u       = k*(xd + xref - x)
der(x)  = -x + u;
```

Solving the system of equations leads to

```
der(x)  := 0
x       := xref
u       := xref
xd      := xref/k
pre(xd) := xd
```

7. Conclusions

This paper has described and illustrated how the new language constructs of Modelica 2 in combination with Modelica's other language elements allow very flexible and powerful initialization conditions to be defined.

Dymola supports Modelica's new way of specifying initial conditions. To support reliable and robust initialization, Dymola manipulates symbolically the initialization problem and generates analytic Jacobians for nonlinear subproblems. Moreover, the special analysis of the initialization problem allows Dymola to

give diagnosis and user guidance when the initialization problem turns out not to be well posed.

Acknowledgements

This work was in parts supported by the European Commission under contract IST-199-11979 with Dynasim AB under the Information Societies Technology as the project entitled "Real-time simulation for design of multi-physics systems".

8. References

[1] D. Brück, H. Elmqvist, S.E. Mattsson, H. Olsson: *Dymola for Multi-Engineering Modeling and Simulation*, Proceedings of Modelica 2002. Modelica homepage: http://www.Modelica.org,

[2] Dymola. *Dynamic Modeling Laboratory*, Dynasim AB, Lund, Sweden, http://www.Dynasim.se

[3] Modelica. *A unified object-oriented language for physical systems modelling - Language Specification.* Version 2.0, Modelica homepage: http://www.Modelica.org,