Robust Initialization of Differential Algebraic Equations

Bernhard Bachmann, Peter Aronsson*, Peter Fritzson+

Dept. Mathematics and Engineering, University of Applied Sciences,
D-33609 Bielefeld, Germany
bernhard.bachmann@fh-bielefeld.de

* MathCore Engineering AB, Teknikringen 1B, SE-583 30 Linköping, Sweden
peter.aronsson@mathcore.com

+ PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
petfr@ida.liu.se

Abstract

This paper describes a new solution method applied to the problem initializing DAEs using the Modelica language. Modelica is primarily an object-oriented equation-based modeling language that allows specification of mathematical models of complex natural or man-made systems. Major features of Modelica are the multidomain modeling capability and the reusability of model components corresponding to physical objects, which allow to build and simulate highly complex systems. However, initializing such models has been quite cumbersome, since initial equations have to be provided at the system level, where the user needs to know details on the underlying transformation and index-reduction algorithms, that in general are applied to simulate a Modelica model.

The new initialization concept allows to define the initial equations locally in each relevant component where the corresponding states appear. This approach also works for arbitrary "well-posed" higher-index problems and makes the initialization of complex systems more user friendly. A prototype implementation in the OpenModelica compiler is presented, and test results of non-trivial application examples are reported.

1 Introduction

So far, using model initialization in Modelica has only been possible for higher-index problems if the user formulates the initial equations globally. This was also the case, e.g. when using the OpenModelica OpenModelica compiler which is an open source implementation developed at PELAB, Linköping University. In order to do such a global formulation successfully, the user needs to know about index reduction, at least the number of freedom left after applying the dummy derivative method is necessary. Therefore, only advanced users have been able to use this feature in the Modelica language, when higher index problems occur (which is very common). In order to provide a more complete simulation environment, we have started to add robust initialization techniques to the OpenModelica compiler.

2 Flattening of a Modelica Model to a Hybrid DAE

A Modelica model is typically translated to a basic mathematical representation in terms of a flat system of *differential and algebraic equations* (DAEs) before being able to simulate the model. This translation process elaborates on the internal model representation by performing analysis and type checking, inheritance and expansion of base classes, modifications and redeclarations, conversion of connect-equations to basic equations, etc. The result of this analysis and translation process is a flat set of equations, including conditional equations, as well as constants, variables, and function definitions. By the term *flat* is meant that the object-oriented structure has been broken down to a flat representation where no trace of the object hierarchy remains apart from dot notation (e.g. `Class.Subclass.variable`) within names.

3 Mathematical Formulation of Hybrid DAEs

3.1 Summary of notation

Below we summarize the notation used in the equations that follow, with time dependencies stated explicitly for all time-dependent variables by the arguments t or t_e:

- $p = \{p_1, p_2, ...\}$, a vector containing the Modelica variables declared as `parameter` or `constant` i.e., variables without any time dependency.
- t, the Modelica variable `time`, the independent variable of type `Real` implicitly occurring in all Modelica models.
- $x(t)$, the vector of state variables of the model, i.e., variables of type `Real` that also appear differentiated, meaning that `der()` is applied to them somewhere in the model.
- $\dot{x}(t)$, the differentiated vector of state variables of the model.
- $u(t)$, a vector of input variables, i.e., not dependent on other variables, of type `Real`. These also belong to the set of algebraic variables since they do not appear differentiated.
- $y(t)$, a vector of Modelica variables of type `Real` which do not fall into any other category. Output variables are included among these, which together with $u(t)$ are algebraic variables since they do not appear differentiated.
- $q(t_e)$, a vector of discrete-time Modelica variables of type `discrete Real`, `Boolean`, `Integer` or `String`. These variables change their value only at event instants, i.e., at points t_e in time.
- $q_{pre}(t_e)$, the values of q immediately before the current event occurred, i.e., at time t_e.
- $c(t_e)$, a vector containing all `Boolean` condition expressions evaluated at the most recent *event* at time t_e. This includes conditions from all if-equations/statements and if-expressions from the original model as well as those generated during the conversion of when-equations and when-statements.
- $rel(v(t)) = rel(cat(1, x, \dot{x}, u, y, \{t\}, q(t_e), q_{pre}(t_e), p))$, a `Boolean` vector valued function containing the relevant elementary relational expressions from the model, excluding relations enclosed by `noEvent()`. The argument $v(t) = \{v_1, v_2, ...\}$ is a vector containing all elements in the vectors $x, \dot{x}, u, y, \{t\}, q(t_e), q_{pre}(t_e), p$. This can be expressed using the Modelica concatenation function `cat` applied to these vectors; $rel(v(t)) = \{v_1 > v_2, v_3 >= 0, v_4<5, v_6<=v_7, v_{12}=133\}$ is one possible example.
- $f(...)$, the function that defines the differential equations $f(...) = 0$ in (1a) of the system of equations.
- $g(...)$, the function that defines the algebraic equations $g(...) = 0$ in (1b) of the system of equations.
- $f_q(...)$, the function that defines the difference equations for the discrete variables $q := f_q(...)$, i.e., (2) in the system of equations.
- $f_e(...)$, the function that defines the event conditions $c := f_e(...)$, i.e., (3) in the system of equations.
- $f_x(...)$, the function that defines the reinitialization values for the continuous variables $x(t_e) := f_x(...)$ at events.

In the context of hybrid DAE:s the *state* of a system is not only made up of the values of the set of variables that occur differentiated in the model. The overall *state* of a system may also include values of discrete variables. In this paper the word *state* is used in this sense, including the state of the discrete part of the system.

3.2 Continuous-Time Behavior

Now we want to formulate the continuous part of the *hybrid DAE* system of equations including discrete variables. This is done by adding a vector $q(t_e)$ of *discrete-time variables* and the corresponding predecessor variable vector $q_{pre}(t_e)$ denoted by `pre(q)` in Modelica. For discrete variables we use t_e instead of t to indicate that such variables may only change value at event time points denoted t_e, i.e., the variables $q(t_e)$ and $q_{pre}(t_e)$ behave as constants between events.

We also make the constant vector p of *parameters and constants* explicit in the equations, and make the time t explicit. The vector $c(t_e)$ of condition expressions, e.g. from the conditions of `if` constructs and `when` constructs, evaluated at the most recent event at time t_e is also included since such conditions are referenced in conditional equations. We obtain the following *continuous DAE* system of equations that describe the system behavior *between* events:

$$\begin{aligned} f(x(t), \dot{x}(t), u(t), y(t), t, q(t_e), q_{pre}(t_e), p, c(t_e)) = 0 \quad (a) \\ g(x(t), u(t), y(t), t, q(t_e), q_{pre}(t_e), p, c(t_e)) = 0 \quad (b) \end{aligned} \tag{1}$$

3.3 Discrete-Time Behavior

Discrete time behavior is closely related to the notion of an event. Events can occur asynchronously, and affect the system one at time, causing a sequence of state transitions.

An event occurs when any of conditions $c(t_e)$ (defined below) of conditional equations changes value from `false` to `true`. We say that an event becomes *enabled* at the time t_e, if and only if, for any sufficiently small value of ε, $c(t_e-\varepsilon)$ is `false` and $c(t_e+\varepsilon)$ is `true`. An enabled event is *fired*, i.e., some behavior associated with the event is executed, often causing a discontinuous state transition.

Firing of an event may cause other conditions to switch from `false` to `true`. In fact, events are fired until a stable

situation is reached when all the condition expressions are `false`.

However, there are also state changes caused by equations defining the values of the *discrete* variables $q(t_e)$, which may change value *only* at events, with event times denoted t_e. Such discrete variables obtain their value at events, e.g. by solving equations in when-equations or evaluating assignments in when-statements. The instantaneous equations defining discrete variables in when-equations are restricted to particularly simple syntactic forms, e.g. *var* `=` *expr*`;`. These restrictions are imposed by the Modelica language in order to easily determine which discrete variables are defined by solving the equations in a when-equation.

Such equations can be directly converted to equations in assignment form, i.e., assignment statements, with fixed causality from the right-hand side to the left-hand side. Regarding algorithmic when-statements that define discrete variables, such definitions are always done through assignments. Therefore we can in both cases express the equations defining discrete variables as *assignments* in the vector equation (1a), where the vector-valued *function* f_q specifies the right-hand side expressions of those *assignments to discrete variable*s.

$$q(t_e) := f_q(x(t_e), \dot{x}(t_e), u(t_e), y(t_e), t_e, q_{pre}(t_e), p, c(t_e)) \quad (2)$$

The last argument $c(t_e)$ is made explicit for convenience. It is strictly speaking not necessary since the expressions in $c(t_e)$ could have been incorporated directly into f_q. The vector $c(t_e)$ contains all `Boolean` condition expressions evaluated at the most recent *event* at time t_e. It is defined by the following vector assignment equation with the right-hand side given by the vector-valued function f_e. This function has as arguments the subset of the discrete variables having `Boolean` type, i.e., $q^B(t_e)$ and $q^B_{pre}(t_e)$, the subset of `Boolean` parameters or constants, p^B, and a vector $rel(v(t))$ evaluated at time t_e, containing the elementary relational expressions from the model. The vector of condition expressions $c(t_e)$ is defined by the following equation in assignment form:

$$c(t_e) := f_e(q^B(t_e), q^B_{pre}(t_e), p^B, rel(v(t_e))) \quad (3)$$

The argument $v(t) = \{v_1, v_2, ...\}$ is a vector containing all scalar elements of the argument vectors. This can be expressed using the Modelica concatenation function `cat` applied to the vectors, e.g. $v(t) = cat(1, x, \dot{x}, u, y, \{t\}, q(t_e), q_{pre}(t_e), p)$. For example, if $rel(v(t))$ = {v_1 > v_2, v_3 >= 0, v_4<5, v_6<=v_7, v_{12}=133} where $v(t)$ = {v_1, v_2, v_3, v_4, v_6, v_7, v_{12}}, then it might be the case that $c(t)$ = {v_1 > v_2 `and` v_3 >= 0, v_{10}, not v_{11}, v_4<5 `or` v_6<=v_7, v_{12}=133}, where v_{10}, v_{11} are `Boolean` variables and v_1, v_2, v_3, v_4, v_6, v_7 might be `Real` variables, whereas v_{12} might be an `Integer` variable. $rel(v(t)) = rel(\texttt{cat}(1, x(t), \dot{x}(t), u(t), y(t), t, q(t_e), q_{pre}(t_e), p))$, is a `Boolean`-typed vector-valued function containing the relevant elementary *relational expressions* from the model, excluding relations enclosed by `noEvent()`.

Discontinuous changes of continuous dynamic variables $x(t)$ can be caused by so-called `reinit` equations in Modelica. As in the case of discrete variables, such discontinuous changes can only occur at events. The effect of a `reinit`-equation that is activated at t_e is an assignment to the continuous variable at time t_e of the form:

$$x(t_e) := f_x(x(t_e), \dot{x}(t_e), u(t_e), y(t_e), t_e, q_{pre}(t_e), p, c(t_e)) \quad (4)$$

For all variables in $x(t_e)$ that are not affected by an `reinit`-equation $f_x(...)$ takes the value of $x(t_e)$, leaving the variable unchanged.

3.4 The Complete Hybrid DAE

The total equation system consisting of the combination of (1), (2), (3) and (4) is the desired *hybrid DAE* equation representation for Modelica models, consisting of *differential*, *algebraic*, and *discrete* equations.

This framework describes a system where the state evolves in two ways: continuously in time by changing the values of the state vector $x(t)$, and instantaneously during events triggered when some of the conditions $c(t_e)$ change value from `false` to `true`. The set of *state variables* from which other variables are computed is selected from the set of differentiated variables $x(t)$, algebraic variables $y(t)$, and discrete-time variables $q(t)$.

4 Simulation of Models Represented by Hybrid DAEs

4.1 Well-defined problem description

A Modelica *simulation problem* in the general case is a Modelica *model* that can be reduced to a hybrid DAE in the form of equations (1), (2), (3) and (4), together with additional constraints on variables and their derivatives called *initial conditions*.

The initial conditions prescribe initial start values of variables and/or their derivatives at simulation time=0 (e.g. expressed by the Modelica `start` attribute value of variables, with the attribute `fixed = true`), or default estimates of start values (the `start` attribute value with `fixed = false`).

The simulation problem is *well defined* provided that the following conditions hold:

- The total model system of equations is consistent and neither underdetermined nor overdetermined.
- The initial conditions are consistent and determine initial values for all variables.
- The model is specific enough to define a unique solution from the start simulation time t0 to some end simulation time t1.

The initial conditions of the simulation problem are often specified interactively by the user in the simulation tool, e.g. through menus and forms, or alternatively as default `start` attribute values in the simulation code. More complex initial conditions can be specified through `initial equation` sections in Modelica.

4.2 Simulation Techniques

There are three different kinds of equation systems resulting from the translation of a Modelica model to a flat set of equations, from the simplest to the most complicated and powerful:

- ODEs – Ordinary differential equations for continuous-time problems.
- DAEs – Differential algebraic equations for continuous-time problems
- Hybrid DAEs – Hybrid differential algebraic equations for mixed continuous-discrete problems.

In the following we present a short overview of methods to solve these kinds of equation systems. However, remember that these representations are strongly interrelated: an ODE is a special case of DAE without algebraic dependencies between states, whereas a DAE is a special case of hybrid DAEs without discrete or conditional equations. We should also point out that in certain cases a Modelica model results in one of the following two forms of purely algebraic equation systems, which can be viewed as DAEs without a differential equation part:

- Linear algebraic equation systems
- Nonlinear algebraic equation systems

However, rather than representing a whole Modelica model, such algebraic equation systems are usually subsystems of the total equation system.

4.3 The Notion of DAE Index

The DAE index is an important property of DAE systems. Consider once more a DAE system on the general form (neglecting the hybrid part, parameters and constants):

$$F(x(t), \dot{x}(t), y(t), u(t)) = 0 \tag{5}$$

We assume that this system is solvable with a continuous solution, given an appropriate initial solution. There are several definitions of DAE *index* in the literature, of which the following, also called *differential index*, is informally defined as follows:

- The index of a DAE system **Error! Reference source not found.** is the minimum number of times certain equations in the DAE must be differentiated in order to solve $\dot{x}(t)$ as a function of $x(t)$, $y(t)$, and $u(t)$, i.e. to transform the problem into ODE explicit state space form.

The index gives a classification of DAEs with respect to their numerical properties and can be seen as a measure of the distance between the DAE and the corresponding ODE

An ODE system on explicit state space form is of index 0 since it is already in the desired form:

$$\dot{x}(t) = f(t, x(t)) \tag{6}$$

The following *semi-explicit* form of DAE system is of index 1 under certain conditions:

$$\begin{aligned} \dot{x}(t) &= f(t, x(t), y(t)) && (a) \\ 0 &= g(t, x(t), y(t)) && (b) \end{aligned} \tag{7}$$

The condition is that the Jacobian of g with respect to y, $(\partial g / \partial y)$ – usually a matrix – is *non-singular* and therefore has a well-defined inverse. This means that in principle $y(t)$ can be solved as a function of $x(t)$ and substituted into (7a) to get state-space form. A DAE system in the general form (5) may have higher index than one. Mechanical models often lead to index 3 DAE systems. We conclude:

- There is no need for symbolic differentiation of equations in a DAE system if it is possible to determine the *highest order derivatives* as continuous functions of time and lower derivatives using stable numerical methods. In this case the index is at most 1.
- The index is zero for such a DAE system if there are no algebraic variables.

4.4 Mixed Symbolic and Numerical Solution of higher-index DAEs

A mixed symbolic and numerical approach to solution of DAEs avoids the problems of numeric differentiation. The DAE is transformed to a lower index problem by using index reduction. The standard mixed symbolic and numeric approach contains the following steps:

1. Use Pantelides algorithm to determine how many times each equation has to be differentiated to reduce the *index* to one or zero.
2. Perform *index reduction* of the DAE by analytic symbolic differentiation of certain equations and by applying the method of dummy derivatives.
3. Select the core state variables to be used for solving the reduced problem. These can either be selected statically during compilation, or in some cases selected dynamically during simulation.
4. Use a numeric ODE solver to solve the reduced problem.

In the following we will discuss the notions of index and index reduction in some more detail.

4.5 Higher Index Problems are Natural in Component-Based Models

The index of a DAE system is not a property of the modeled system but the *property* of a *particular model representation*, and therefore a function of the modeling methodology. A natural object-oriented component-based methodology with reuse and connections between physical objects leads to high index in the general case. The reason is the constraint equations resulting from setting variables equal across connections between separate objects.

Since the index is not a property of the modeled system it is possible to reduce the index by symbolic manipulations. High index indicates that the model has algebraic relations between differentiated state variables implied by algebraic relations between those state variables. By using knowledge about the particular modeling domain it is often possible to manually eliminate a number of differentiated variables, and thus reduce the index. However, this violates the object-oriented component-based modeling methodology for physical modeling that is intended to be supported by the Modelica language.

We conclude that high index models are natural, and that automatic index reduction is necessary to support a general object-oriented component-based modeling methodology with a high degree of reuse.

5 Finding Consistent Initial Values at Start or Restart

As we have stated briefly above, at the start of the simulation, or at restart after handling an event, it is required to find a consistent set of initial values or restart values of the variables of the hybrid DAE equation system before starting continuous DAE solution process.

At the *start* of the simulation these conditions are given by the initial conditions of the problems (including `start` attribute equations, equations in `initial equation` sections, etc., together with the system of equations defined by (1), (2), and (3). The user specifies the initial time of the simulation, t_0, and initial values or guesses of initial values of some of the continuous variables, derivatives, and discrete-time variables so that the algebraic part of the equation system can be solved at the initial time $t=t_0$ for all the remaining unknown initial values. In some application examples it is even necessary to calculate initial values of parameters (`fixed = false`), that afterwards be kept constant during simulation.

At *restart* after an event, the conditions are given by the *new values* of variables that have changed at the event, together with the current values of the remaining variables, and the system of equations (5), (6), and (7). The goal is the same as in the initial case, to solve for the new values of the remaining variables. In the initial case, however, the causality can be different since initial equations are included to calculate start values for the state variables, whereas at restart the state variables are always known.

6 Robust Initialization of Higher-Index DAEs

Initializing DAEs using the Modelica language has been quite cumbersome in the past, since initial equations have to be provided on the system level, where the user needs to know details on the underlying transformation and index-reduction algorithms, that are in general applied to simulate a Modelica model. Especially, when higher-index DAEs are involved the number of locally defined state variables no longer coincide with the number of state variables of the overall system. Although, one can influence the index-reduction algorithm by setting some attribute values (`stateSelect=always,prefer,…`), cases can be constructed which don't allow the straight forward prediction of the number of state variables left after transformation.

In order to make the initialization procedure more convenient a new concept is necessary, which allows to define the initial equations locally in each relevant component where the corresponding states appear, even if these states are eliminated during index-reduction. Naturally, this leads to an overdetermined system of equations, which has to be solved during the initialization process. In this context, we call a higher-index problem "well-posed" if enough equations of the system are redundant so that initial values can be determined which fulfill the whole set of initial equations. The main idea of the new approach is to reformulate

the problem of finding roots of the set of non-linear equations to an equivalent optimization problem.

Considering the general mathematical description of the initialization problem:

$$\begin{aligned} f_1(z_1,\ldots,z_n) &= 0 \\ &\vdots \\ f_m(z_1,\ldots,z_n) &= 0 \end{aligned} \qquad (8)$$

Cases where $m \geq n$ means that more equations (m) than variables (n) are given. Every solution to (8) minimizes the problem:

$$F(z_1,\ldots,z_n) = \sum_{i=1}^{m} f_i(z_1,\ldots,z_n)^2 \rightarrow \min \qquad (9)$$

On the other hand, every global minimum of (9) is a solution to (8). In order to solve (9) a number of different algorithms have been developed during the past. The algorithm can be categorized depending on the order of derivatives needed during the solution process. In the OpenModelica environment the Simplex-method of Nelder and Mead as well as the Brent's method are currently implemented, only working with the minimization function F. The OpenModelica prototype already shows reliable results for the evaluated examples.

Further improvements can be achieved as soon as the Jacobian of F with regards to the unknown is available. In that case, more advanced algorithms like the method of Fletcher-Reeves, Quasi-Newton, and/or Levenberg-Marquardt methods can be applied which would provide a speed-up in convergence. We regard this as a quality of implementation, since the described approach is working in principle already.

7 Test and Evaluation with OpenModelica

Consider the following electrical 3-phase power system, where two generating units `VS1` and `VS2` are connected via a transmission line modeled by components LR1 and LR2.

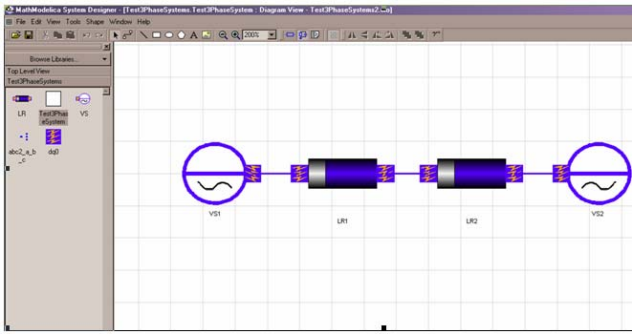

Figure 1. An electrical power system where two generating units `vs1` and `vs2` are connected via a transmission line.

The connectors are written in `dq0`-coordinates implementing the potential variable `u_dq0` and the flow variable `i_dq0`. These quantities are constant in case of a nondistributed steady state, which is generally assumed during the initialization process. Introducing the Park-Transformation P the 3-phase rotating system (voltages `u_abc` and currents `i_abc`) can be calculated from the `dq0`-representation and vice versa.

The transmission line (`LR1` and `LR2`) is modeled by a purely inductive and resistive component, based on the Modelica Electrical Library. Since `LR1` and `LR2` are connected in series, giving a higher index system, index reduction has to be applied for simulation purposes.

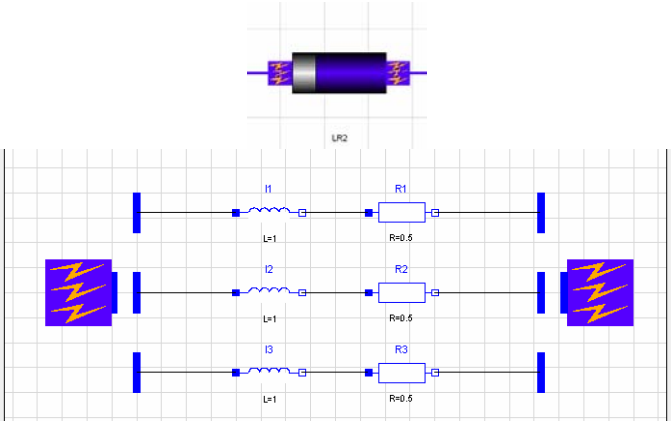

Figure 2. LR2 component with dq0 connectors.

The voltage source is described similarly using the Modelica Standard Library combined with the `dq0`-connectors.

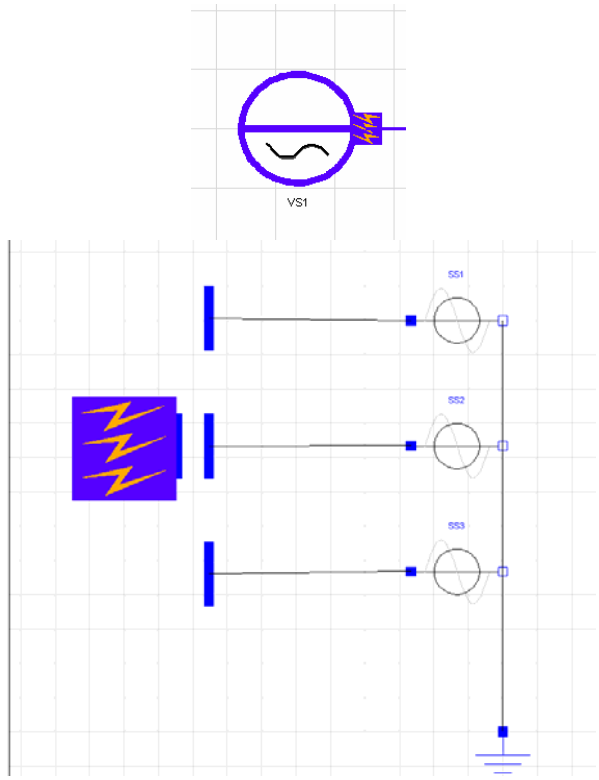

Figure 3. Voltage source.

In order to initialize the model correctly to steady state the following initial equations have been added to the local components `LR1` and `LR2`.

```
model LR
  ...
equation
  ...
initial equation
  der(dq0_1.i_dq0)={0,0,0};
end LR;
```

Due to the higher-index of the overall system, index-reduction is applied. The system finally is determined by 3 state variables `LR1.I1.i`, `LR1.I2.i`, `LR1.I3.i`. The corresponding initial equation system has 3 equations more than number of unknowns, but these equations are redundant and could be eliminated. Due to the involvement of the Park-transformation, redundancy is not easy to detect. However, applying the concept described above correct initialization of the system is performed.

8 Implementation Status

An experimental prototype version of this method has been implemented in a special version of the OpenModelica compiler (not yet in the ordinary version), and tried on several small examples. We have worked for some time to automatically handle the example described in this paper have been delayed by a bug in the OpenModelica index reduction. The example has been verified by partly manual efforts. However, we expect to soon fix this small remaining problem in the OpenModelica compiler.

9 Conclusions and Future work

In this paper we have presented an overview of our implementation of initializing Modelica models in the OpenModelica compiler. A new concept has been developed to describe the initial equations locally in the relevant component where the corresponding states appear, that also works for arbitrary well-posed higher-index problems. Due to the necessary index reduction some of the states get changed to dummy states that means that they will be algebraic during the simulation of the model. The corresponding initial equations are therefore redundant, but can be handled correctly by the new initialization process, if they are consistent. If not, an error/warning is issued to the user.

The implementation is however not yet complete. The current prototype just implements the concept, but the efficiency should be increased in the near future. We wish to implement calculation of the Jacobian matrix of the equation system with regards to the state variables. This gives the possibility to implement more advanced and robust numerical algorithms in order to solve the corresponding optimization (minimization) problem during initialization of the DAE.

10 Acknowledgements

This work was supported by the University of Applied Sciences in Bielefeld, by MathCore Engineering AB, by the Swedish Research Council (VR), and by SSF in the VISIMOD project.

References

[1] Peter Fritzson, et al. The Open Source Modelica Project. In Proceedings of The 2nd International Modelica Conference, 18-19 March, 2002. Munich, Germany See also: http://www.ida.liu.se/projects/OpenModelica.

[2] Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pp., ISBN 0-471-471631, Wiley-IEEE Press, 2004.

[3] The Modelica Association. The Modelica Language Specification Version 2.2, March 2005. http://www.modelica.org.

[4] The OpenModelica Users Guide, version 0.6, June 2005. www.ida.liu.se/projects/OpenModelica

[5] The OpenModelica System Documentation, version 0.6, June 2006.
www.ida.liu.se/projects/OpenModelica

[6] K. E. Brenan, S. L. Campbell, and L. R. Petzold, Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations, Elsevier, New York, 1989.

[7] B. Bachmann et. al. (Modelica Association): Modelica - A Unified Object-Oriented Language for Physical Systems Modeling - Language Specification. 2002.

[8] P. Fritzson, P. Aronsson, P. Bunus, V. Engelson, L. Saldamli, H. Johansson, A. Karström: The Open Source Modelica Project. In: 2nd Modelica Conference 2002, Oberpfaffenhofen, 2002

[9] S.-E. Mattson, H. Olson, H. Elmqvist: Dynamic Selection of States in Dymola. In: 1st Modelica Workshop 2000, Lund, Sweden, 2000

[10] M. Otter: Objektorientierte Modellierung Physikalischer Systeme (Teil 4) – Transformationsalgorithmen. In: at Automatisierungstechnik, Oldenbourg Verlag München, 1999

[11] M. Otter, B. Bachmann: Objektorientierte Modellierung Physikalischer Systeme (Teil 5,6) – Singuläre Systeme. In: at Automatisierungstechnik, Oldenbourg Verlag München, 1999

[12] R. Fletcher: Practical Methods of Optimization John Wiley & Sons, 1995

[13] J. Stoer, R. Burlisch: Einführung in die numerische Mathematik. Springer Verlag, 1994

[14] S.E. Mattsson, G. Söderlind: Index reduction in differential-algebraic equations using dummy derivatives. SIAM Journal of Scientific and Statistical Computing, Vol. 14, 1993.

[15] K.E. Brenan., S.L. Campbell, L.R. Petzold: Numerical Solution of Initial Value Problems in Differential Algebraic Equations. North-Holland, Amsterdam, 1989

[16] C.C. Pantelides: The Consistent Initialization of Differential-Algebraic Systems, SIAM Journal of Scientific and Statistical Computing, 1988.

[17] L.R. Petzold: A description of DASSL: A differential / algebraic system solver. Sandia National Laboratories, Albuquerque, 1982

[18] H. Elmqvist: A Structured Model Language for Large Continuous Systems, PhD dissertation, Department of Automatic Control, Lund Institute of Technology, Lund, Schweden, 1978

[19] R.E. Tarjan: Depth First Search and Linear Graph Algorithms. SIAM Journal of Comp., Nr. 1, 1972.