Modeling and simulation of differential equations in Scicos

Masoud Najafi Ramine Nikoukhah
INRIA-Rocquencourt, Domaine de Voluceau,
78153, Le Chesnay Cedex France

Abstract

Block diagram method is an old approach for the modeling and simulation of differential equations. Modeling and simulation of some kind of differential equations such as differential-algebraic equations (DAE) is cumbersome, difficult, or even impossible with this approach. Scicos which is a modeling and simulation software based on Block diagram approach has recently been developed to simulate Modelica programs. In this paper, it will be explained the way different classes of DAE can easily be specified in Modelica and simulated in Scicos.
Keywords: hybrid differential equations; numerical solver; simulation; Modelica; Scicos

1 Introduction

Scilab[1] is a free, and open-source software for scientific calculation. Scicos[2] is a toolbox of Scilab and provides an environment for modeling and simulation of dynamical systems [6, 4]. For many applications, the Scilab/Scicos environment provides an open-source alternative to Matlab/Simulink and MatrixX [14, 15]. Scicos includes a graphical editor for constructing models by interconnecting blocks, representing predefined or user defined functions, a compiler, a simulator, and code generation facilities. A Scicos block diagram is composed of blocks and connection links. A block corresponds to an operation and by interconnecting blocks through links, we can construct a model, or an algorithm.
The Scicos blocks represent elementary systems that can be used as model building blocks. They can have several inputs and outputs, continuous-time states, discrete-time states, zero-crossing functions, etc. Scicos allows customization with regard to incorporating user *C*, *Fortran*, or Scilab codes. Scicos translates the block diagram model into a system of Ordinary Differential Equations (ODE) or Differential Algebraic Equation (DAE) and applies an ODE or a DAE solver in order to perform a simulation. A block diagram system representation can only be used to model ODEs and a special class of DAEs, while the solvers used in Scicos support a larger class of DAEs [9]. In this paper we will explain the way Scicos environment has been developed to write and simulate a large class of differential equations, called hybrid differential equations. To get an idea what a Scicos model looks like, a model of a simple control system implemented in Scicos has been shown in Figure 1. In Figure 1 the `Clock` block

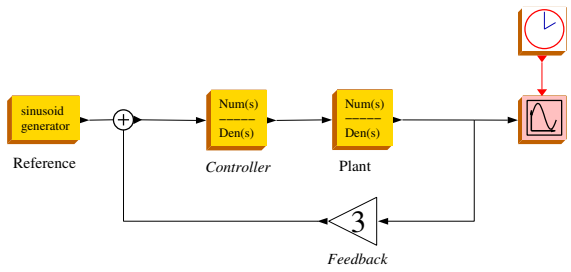

Figure 1: A Scicos model of a control system

generates a periodic activation signal (event) that activates the `Scope` block. At event times the scope reads its input signal and displays them.

2 Mathematical Background

A differential equation expressed either by an *Ordinary Differential Equations* (ODE), *i.e.*,

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

where $\dot{x}$ denotes the derivative of x, the state variables, with respect to the time variable t, and u is the input vector variable, or by *Differential Algebraic Equations* (DAE) [2, 3, 5], *i.e.*,

$$\begin{cases} \dot{x} = f(x,y,u,t) \\ 0 = g(x,y,u,t) \end{cases} \quad (1)$$

[1] `www.scilab.org`
[2] `www.scicos.org`

where (1-a) is the differential part and (1-b) is the algebraic part of the DAE. The equation set (1) is a *semi-explicit* DAE, where the differential and the algebraic parts are decomposed. If we cast (1) in the form of

$$0 = F(\dot{z}, z, t), \quad z = \begin{pmatrix} x \\ y \end{pmatrix} \tag{2}$$

This system is called a *fully implicit* DAE. Note that if $\frac{\partial F}{\partial \dot{z}}$ is non-singular, then it is possible to formally solve $\dot{z}$ as a function of z in order to obtain an ODE. However, if it is singular, this is no longer possible and the solution z has to satisfy certain algebraic constraints.

DAEs are characterized by their index. The *index* of a DAE , *e.g.*,(2), is the smallest number of differentiation of (2) to obtain an ODE by algebraic manipulations [7]. In general, the higher the index, the greater the numerical difficulty one encounters, when trying to integrate the DAE numerically. In a semi-explicit index-1 DAE (1), variables whose derivatives appear in DAE are called *differential variables* and the other ones are called *algebraic*, *i.e.*, x in (1-a) is differential and y in (1-b) is algebraic [16, 17].

2.1 Numerical solvers of Scicos

In order to integrate differential equations or simulate any model, Scicos uses two numerical solvers; `Lsodar` [8, 16] and DASKR [16, 2]. `Lsodar` is an ODE solver which is used when the Scicos diagram represents an ODE. If a diagram represents a DAE, DASKR is used. DASKR is a variable step, variable order index-1 DAE solver. The solver properties discussed here are those of DASKR and LSODAR; however these properties are common to most modern solvers. The most important of these properties will be explained in the following subsections [11].

Consistent initial condition: Most of the problems in using standard solvers are common to both ODE and DAE solvers. However, there is an additional difficulty with the DAE case: the problem of re-initialization and finding consistent initial conditions. Simulation of an ODE can be started from any initial state, but simulation of a DAE should be started from a consistent initial state. This is an additional difficulty in simulation of DAEs.

Continuity criteria for numerical solver: DASKR and LSODAR use a variable order variable step-size BDF (backward differentiation formula) method to integrate. The BDF methods normally need continuity in variables and their derivatives [1]. Consequently, DASKR and LSODAR require that the system be sufficiently smooth over an integration period. This means that simulator must make sure to stop and reinitialize the solver at each potential point of non-smoothness (discontinuity, discontinuity in the derivative, etc.) of the ODE/DAE. These ODE/DAEs require some additional solver features, such as event detection or root finding.

Event detection: The ability to detect the time when a discontinuity occurs, or more precisely, the time when a function crosses some given value (by default considered zero) is of capital importance in simulation of DAEs. For this purpose, the discontinuity function is given to the solver as a zero-crossing function. When a zero-crossing occurs, the solver stops the integration and returns the exact crossing time to the main program. So, it is important to halt the solver and restart at the discontinuity point. The numerical solver can also provide the direction in which a function has crossed the zero.

2.2 Discontinuity handling in the simulator of Scicos

DAEs may have discontinuities or may be variable structure or may change at certain points in time. Such types of DAEs are called hybrid DAEs. A hybrid DAE is a way of describing non-smooth multi-model systems in terms of a finite number of smooth systems. The idea is to divide the state space of the system into different regions. It is assumed that the system is described in terms of a single smooth DAE within each region. A simple example of a multi-model DAE is:

$$\begin{array}{llll} \texttt{If} & (g(x) > 0) & \texttt{then} & f_1(\dot{x}, x, u, t) = 0 \\ & & \texttt{else} & f_2(\dot{x}, x, u, t) = 0 \end{array} \tag{3}$$

where the DAE has two models: the first one is on when $(g(x) > 0)$ and the second is when the condition is not true. This switching may cause a discontinuity in the signals, so they cannot be integrated by the numerical solvers. In order to cope with this problem, the discontinuity should be detected and the solver be reinitialized after the discontinuity point. To detect and localize the discontinuity time, solvers use zero-crossing functions that cross zero over the discontinuity point. For example, for (3), we use $g(x)$ as the zero-crossing function [11].

For localizing the discontinuity point, the simulator associates `Mode` variables with each zero-crossing functions in Scicos. Mode variables are used to assign and fix an ODE/DAE in every time interval between

each two discontinuities. In general, when the numerical solver is called, the system of equation should not be changed. During integration, the zero-crossing functions that indicate the conditions for the model change are examined by the solver. In case of any zero-crossing, the Mode variables should be updated to feed another ODE/DAE to the solver. It should be note that Mode variables are defined in the Scicos simulator and is transparent to the user. In Scicos, a Mode variable is assigned systematically to each discontinuity point, characterized by an `If-then-Else` block. Scicos considers the system (3) in the following form:

$$0 = \begin{cases} f_1(\dot{x},x,u,t) & \texttt{if} \quad \texttt{Mode} = 1 \\ f_2(\dot{x},x,u,t) & \texttt{if} \quad \texttt{Mode} = 2 \end{cases}$$

$$\begin{array}{llll} \texttt{when} & (g(x) \geq 0) & \texttt{then} & \texttt{Mode} = 1 \\ \texttt{when} & (g(x) < 0) & \texttt{then} & \texttt{Mode} = 2 \end{array}$$

By default, for any discontinuity in the model, a Mode is used. But it should be noted that any `If-then-Else`, not resulting in a discontinuity, can be used without Mode. If the resulting `if-then-else` expression is smooth, the modeler has the possibility to give this extra information to the simulator in order to avoid these unnecessary solver reinitialization. That is why there is a parameter in `If-then-Else` blocks that lets the user define whether the block is used with or without zero-crossing. The Mode variables should not be used in some cases. For example, when a function is not defined everywhere and might be called near the limit of validity. In DAE (4),

$$\begin{aligned} \dot{x} &= -xy - x + y \\ y &= \begin{cases} -1+\sqrt{x} & \texttt{if } x \geq 0 \\ -1+\sqrt{-x} & \texttt{if } x < 0 \end{cases} \end{aligned} \tag{4}$$

if the Mode variable is used the first model (*i.e.*, $\sqrt{x}$) is employed until the discontinuity point $x = 0$ is detected. During the search process the solver uses $\sqrt{x}$ for $x < 0$ to localize the crossing point. This will cause a failure in the integration, so the Mode variable should not be used to permit the solver probe beyond the discontinuity point.

3 Implementing differential equations with block diagram approach

Block diagram implementation is an old method to represent differential equations. In this method, through the use of multipliers, adders, integrators, etc. a differential equations is constructed graphically. Block diagram consists of blocks that are connected by arrows and each block is a transducer that transforms the incoming signals to one or more output signals. A block can represent simple arithmetic operations or functions without memory, but also operations whose results are dependent on previous inputs to the block, *i.e.*, with memory. Several software tools such as Scicos, Simulink, SystemBuild, etc. use block diagram method to model and simulate dynamical systems. As an example the diagram in Figure 2 displays the graphical representation of equation (5).

$$\begin{cases} \dot{x}_1 & = -0.04x_1 + 10^4 x_2 x_3 \\ \dot{x}_2 & = 0.04x_1 - 10^4 x_2 x_3 - 3\times 10^7 x_2^2 \\ \texttt{if} & (1+\sin(0.1t) - x_2 - x_1 > 0.5) \\ & \texttt{then } x_3 = -10x_1, \\ & \texttt{else } x_3 = 10x_1. \end{cases} \tag{5}$$

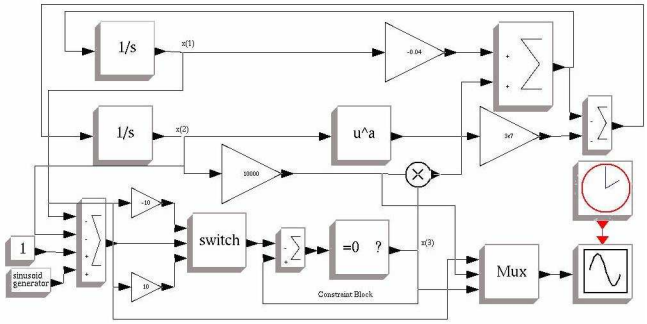

Figure 2: Block diagram implementation of DAE (5) in Scicos

3.1 Shortcomings in the block diagram approach

It is often possible to model differential equations via block diagram approach, but in fact it is not an easy and efficient way. There are several shortcomings. In a block diagram model a small change in the differential equations follows with another study to rearrange the entire structure of the block diagram which may have little similarity to the previous version. Furthermore, most of the general-purpose simulation softwares on the market such as ACSL, Simulink and SystemBuild that use block diagram approach assume that a system can be decomposed into block diagram structures with causal interactions. This means that the models should be expressed as an interconnection of models on semi-explicit form, *i.e.*, $M(t)\dot{x} = f(x,t)$, where the matrix $M(t)$ is singular [18]. Although theoretically any DAE index-1, can be transformed into a semi-explicit DAE,

since such a transformation is done manually, it is time consuming and sometimes it is practically impossible.

With these shortcomings, there is a rising need to have an appropriate framework for DAE representation. A convenient way is to work with the DAEs as the text. The reader may think of simulation methods in which DAEs are expressed in a textual environment, such as writing computer programs and invoking the numerical solvers, or writing Scilab or Matlab script files. But these methods are not efficient and do not provide a proper framework to control and interact with the numerical solver. In fact, the numerical solver considers the DAE as a black box and the internal discontinuities remain hidden. For example, if a discontinuous DAE is simulated directly by `Dassl` function in Scilab or by `ode15s` function in Matlab, the simulation would fail. As an example, when we tested DAE (5) with Matlab, the simulation failed and the following error message raised:

```
>> Warning:  Failure at t=5.235698e+00.
Unable to meet integration tolerances
without reducing the step size
below the smallest value allowed
(1.860093e-14) at time t.
```

The problem lies in the solver control, *i.e.*, a discontinuous DAE cannot be integrated by the numerical solver without discontinuity handling and a good restart managements. Scicos has recently been developed to simulate non-casual models and the user can write physical models symbolically with a new the Modelica language [10].

4 Modelica language

Modelica[3] is a freely available, object oriented, general purpose language for modeling of physical systems, *e.g.*, mechanical, electrical and control systems. The Modelica language allows a direct and convenient specification of systems with continuous-time and discrete-time dynamics. Although Modelica is a rich language having the capacity to handle continuous-time and discrete-time behaviors, in this paper we will focus mainly on modeling hybrid differential equations. A Modelica program or model like any other computer language is composed of a variable or component declaration section and an equation section. Suppose that we want to model DAE (6) in Modelica.

$$\begin{cases} \dot{x} = x - xy \\ \dot{y} = yx - 2y \\ x(0) = 1 \\ y(0) = 2 \end{cases} \tag{6}$$

This DAE consists of two differential variables, *i.e.*, x and y. They are continuous-time `Real` type variables and their initial value at beginning (time=0) are `1` and `2`, respectively. Here is the Modelica program:

```
class Oscillator "Oscillator model"
  Real x(start =1), y(start=2);
equation
  der(x) = x-x*y;
  der(y) = x*y-2*y;
end Oscillator;
```

In the first part of the program, two variables and their initial values are declared. The next part of the program contains the equations. In this section, there are two equations for two unknowns, *i.e.*, `der(x)` and `der(y)` the time derivatives of `x` and `y`. In Modelica, equations are composed of expressions both on the left hand side and the right hand side. It is neither required to write the equations in form of assignments, nor to write the equations in a specified order. It is, however, important to provide equal number of unknowns and equations. For instance, here is the above program that has been rewritten without changing the model mathematically:

```
class Oscillator2 "Oscillator model"
  Real x(start =1), y(start=2), v;
equation
  der(x)-x+v=0;
  0=-der(y)-2*y+ v;
  x*y=v;
end Oscillator2;
```

Note that in this program, `v` is an algebraic variable. In Modelica the initial value of all variables can be specified. Theoretically specifying initial value of algebraic states is not required. This, however, would help the numerical solver to find the consistent initial condition for highly nonlinear DAEs or in case where there are several solutions it acts as a guess value to help the solver to catch the desired solution [13].

In Modelica models that are in fact the mathematical equations, it is not possible to classify (at least a-priori) the variables as inputs and outputs. This type of models are called *acausal* models. That is in contrast with *causal* models where there are explicit inputs and outputs and the outputs are computed as function of inputs and other internal variables. To make an analogy

[3] www.Modelica.org

with computer programming languages, causal models correspond to the use of assignment statements where the right-hand sides of the equations are evaluated and the result of the evaluation is assigned to the variables on the left-hand side of the equations [10].
An acausal model cannot be simulated directly, it should be transformed into a causal model. In general, it is possible to convert an acausal model into a causal model by rewriting the equations and finding the appropriate causality structure in equations. In Scicos this is done by the Modelica compiler. The Modelica compiler receives the Modelica program and extracts the necessary information for the numerical simulation and generates a usable *C* program for Scicos.

5 Hybrid DAE modeling in Modelica

In block diagram approach one cannot model fully-implicit DAEs directly. In Modelica this constraint does not exist and any DAE[4] can be expressed without making any effort to transform them into an explicit form. A multi-model DAE or a DAE with discontinuity is defined with `If-then-else` constructs. Note that an `If` should always be used with an `else`. As an example, a Modelica Code for the DAE (5) follows:

```
class DAE2
   Real x1(start=1.0), x2 (start=0.0), x3,xs;
equation
   der(x1) = -0.04*x1 + 1e4*x2*x3;
   der(x2) = 0.04*x1 - 1e4*x2*x3 - 3e7*x2*x2;
   x3 = if (xs>0.5) then -10*x1 else 10*x1;
   xs = 1 + sin(0.1*time)-x2-x1;
end DAE2;
```

For this DAE, the Modelica compiler automatically extracts `(xs-0.5)` as the discontinuity or zero-crossing function and assigns a Mode variable during the generation of the *C* program. `If-then-else` constructs can also be used to define multi-model DAEs. For example, for the following multi-model DAE

$$\texttt{if } (x \geq 0) \texttt{ then} \begin{cases} 0 = \dot{x}^3 - xy\dot{x} - x + y^2 + 1 \\ 0 = \dot{y}\dot{x} + yx - x \end{cases}$$

$$\texttt{else} \begin{cases} 0 = \dot{y} - 2y\dot{x} + y + x^2 - 1 \\ 0 = 5\dot{x} + 2 - 2yx + \dot{y}x + sin(t) \end{cases}$$

we can write this Modelica code:

```
0=if (x>=0) then  der(x)^3-x*y*der(x)-x+y*y+1
  else  der(y)-2*y*der(x)+y+x*x-1;

0=if (x>=0) then  der(y)*der(x)+y*x-x
  else  der(x)*5+2-2*y*x+der(y)*x+sin(time);
```

[4] In the current version of the Modelica compiler of Scicos only index-1 DAEs are accepted

By default, the Modelica compiler of Scicos associate Mode variables with discontinuity points. When a discontinuity does not need any special treatments, the compiler should be informed with `noEvent()` operator. For example, for DAE (4) we write

```
der(x)=-y*x-x+y;
y=if noEvent(x>=0) then -1+sqrt(x)
                   else -1+sqrt(-x);
```

In this case, the Modelica compiler does not consider the condition as a zero-crossing functions and during the simulation, the solver does not stop at `x=0`. For `(x>0)`, $\sqrt{x}$ is evaluated and for `(x<0)`, $\sqrt{-x}$ is used. In general, `noEvent()` performs two things: First, it inhibits the solver to probe for solutions beyond the limit of validity. Then, it prevents the solver from halting the integration and doing an unnecessary restart.

Modelica can also be used for mixed continuous-time and discrete equations. For the discrete-time parts, the synchronous data flow principle with the single assignment rule is used. Discrete event and discrete-time models are supported by `when` statements. The equations in a `when` clause are conditionally activated at instants (called event) where the `when` condition becomes true. Here is an example to show the way a discrete-time equation is written in Modelica. The difference equation should be updated whenever x crossed zero with a positive to negative direction, *i.e.*,

$$\begin{cases} \ddot{x} = -4x \\ \texttt{update } z(k+1) = 0.9\, z(k) - 0.2 \end{cases}$$

we can write the following modelica program.

```
class Sine
  Real x(start=1), y(start=0);
  discrete Real z(start=3), z1(start=-1);
equation
    der(x)=y;
    der(y)=-4*x;
    when (x<0) then
      z=0.9*z-0.2;
    end  when;
end Sine;
```

The Modelica compiler deduces the direction of the zero-crossing form the condition `(x<0)`. Because this condition becomes true when the `x` becomes negative. So far, Scicos provides a minimum support for Modelica discrete models. That is because the discrete time models can be modeled in the Scicos environment. It is however envisaged to improve Modelica compiler of Scicos to support Modelica discrete models.
In `when` clauses, continuous-time variable can also be initialized. A special operator `reinit(state,`

NewValue) can be used to assign new values to the continuous states of a model at an event time. reinit can only be employed in the body of a when-clause. As an example, consider the bouncing ball system. Whenever the ball hits ground, *i.e.*, its height becomes negative, the velocity changes sign and dampens down. Here is a Modelica code for this hybrid system.

```
class Bounce
  Real y(start=10), v(start=0);
equation
  der(y) = v;
  der(v) = -9.8;
  when y < 0 then
    reinit(v, -0.9*v);
  end when;
end Bounce;
```

6 Simulation of Modelica programs

Modelica is a language that provides an environment to express the differential and algebraic equations. Note however that the main goal is simulating the models. In order to simulate Modelica models, they should be transformed into a causal program. For that, the Modelica models should be compiled. There are several Modelica compilers such as Dymola[5] and Open-Modelica[6]. Scicos has its own Modelica compiler called Modelicac (acronym of "Modelica compiler") for a subset of the Modelica language. Modelicac is an external tool, *i.e.*, it is independent of Scilab. By default, Modelicac comes with a module that generates a *C* code for Scicos blocks. However, since Modelicac is free and open source, it is possible to develop code generators for other targets as well.

A Modelica program is associated with a Scicos block. A Scicos block whose behavior is written in Modelica is called an implicit block [10]. With the associated implicit block the input/output variables of the Modelica program can be defined or visualized. This block may be connected to other blocks to build a bigger model, see for example Figure 3 in which a simple electrical circuit has been built with implicit blocks and some output variables are visualized with ordinary or explicit blocks..

The Modelica compiler uses the input and output variables to establish a causality between the variables in the Modelica program. In the next stage, the compiler

[5] www.dymola.com
[6] www.modelica.org

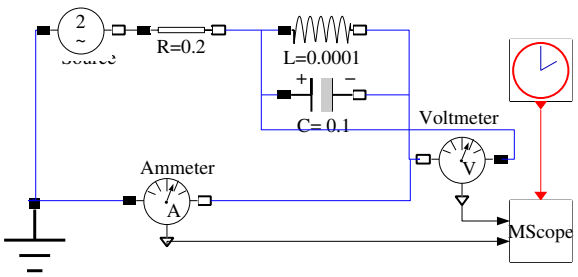

Figure 3: A Modelica block for Oscillator3.mo

simplifies the equations and eliminates the unnecessary variables if possible. In the final stage a *C* program that has the input/output behavior of the Modelica program is generated [13]. Most of the time, the simplification and elimination of variables reduces the size of DAE that consequently reduces the integration time. In addition, a semi-explicit DAE form may be obtained that simplifies the numerical integration [12]. In order to demonstrate the simulation of a complete example, consider this Modelica program

```
class Oscillator3 "Oscillator model"
  Real x(start =1), y(start=2), u;
equation
  der(x) = x-x*y;
  der(y) = x*y-u*y;
end Oscillator3;
```

where u is unknown and is defined by user or another block. To simulate Oscillator3, we use a Modelica block (see Fig. 4) . In the dialog box of the block (see Fig. 5) an input variable u and two output variables x, y are defined. After clicking on OK, another window lets the user write the program (see Figure 6).

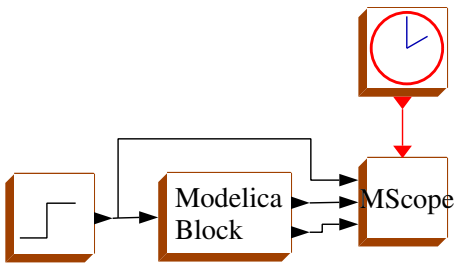

Figure 4: A Modelica block for Oscillator3.mo

When the program is compiled, a *C* program is generated. Here is a fragment of the generated code.

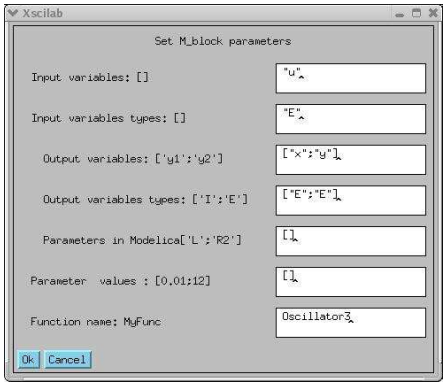

Figure 5: Defining the Modelica program input/output variables

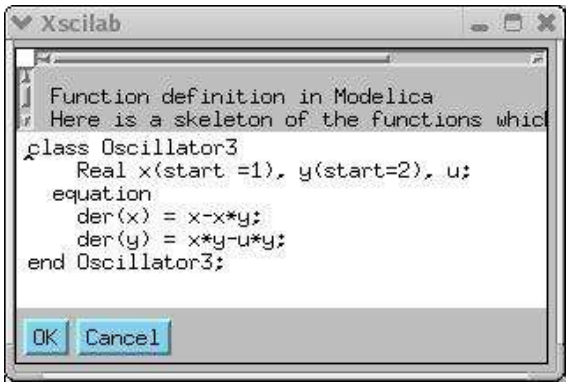

Figure 6: Modelica program in a Modelica block

```
if (flag == 0) { // generated DAE code
   res[0] = xd[0]+x[0]*x[1]-x[0];
   res[1] = u[0]*x[1]+xd[1]-x[0]*x[1];
}else if (flag == 1) {  //output update
   y[0][0] = x[0];
   y[1][0] = x[1];
}else if (flag == 4) {// initial values
   x[0] = 1.0;
   x[1] = 2.0;
}
```

The simulation result is given in Figure 7.
As another example, consider the DAE (5) whose block diagram implementation is depicted in Figure 2. In this case, the Modelica block has only three outputs, see Figure 8-10. The simulation result is given in Figure 11.

7 Conclusion

Modeling and simulation of DAEs via block diagram approach has several shortcomings. Scicos which is a simulation software based on block approach has recently been developed to provide another approach for

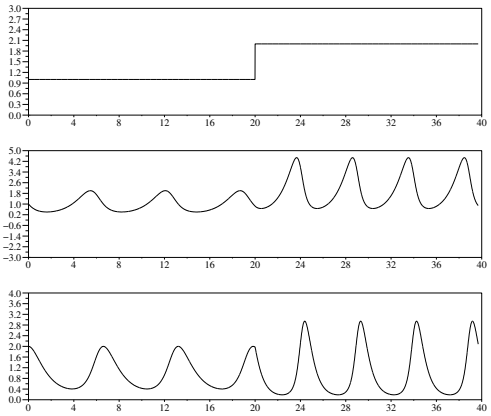

Figure 7: Simulation results for the Scicos diagram in Figure 4

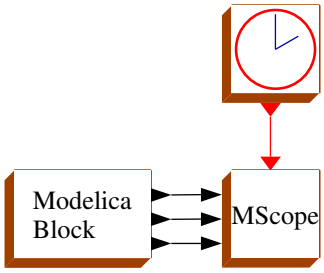

Figure 8: A Modelica block for the DAE (5)

modeling DAEs, *i.e.*, using the Modelica language. In this paper, with some examples we explained the way hybrid DAEs are simulated in Scicos. The Modelica language and its use in Scicos in modeling and simulation of hybrid DAEs were explained. In the last part of the paper, a simple Scicos block is introduced to write and simulate Modelica programs in Scicos.

References

[1] BRENAN, K. E., CAMPBELL, S. L., AND PETZOLD, L. R. Numerical solution of initial-value problems in differential-algebraic equations. *SIAM pubs., Philadelphia* (1996).

[2] BROWN, P. N., HINDMARSH, A. C., AND PETZOLD, L. R. Consistent initial condition calculation for differential-algebraic systems. *SIAM Journal on Scientific Computing 19*, 5 (1998), 1495–1512.

[3] CAMPBELL, S. L. Numerical methods for unstructured higher index daes. *Annals of Numeri-*

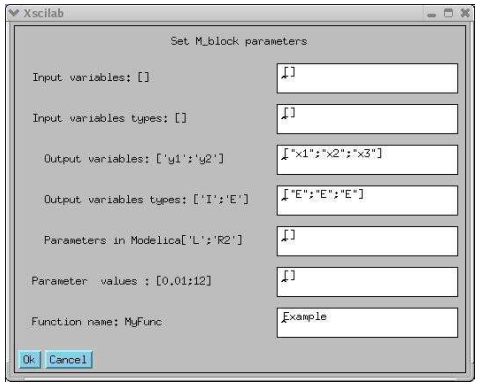

Figure 9: Defining the Modelica program input/output variables

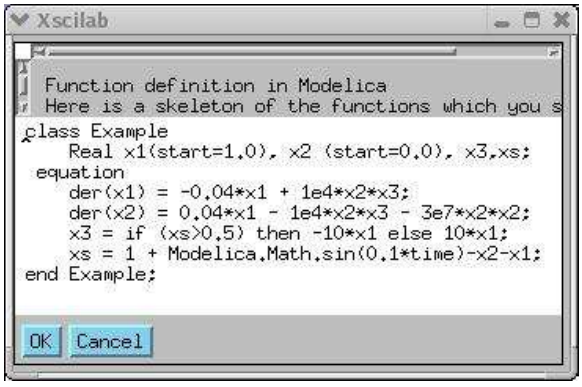

Figure 10: Modelica program in a Modelica block

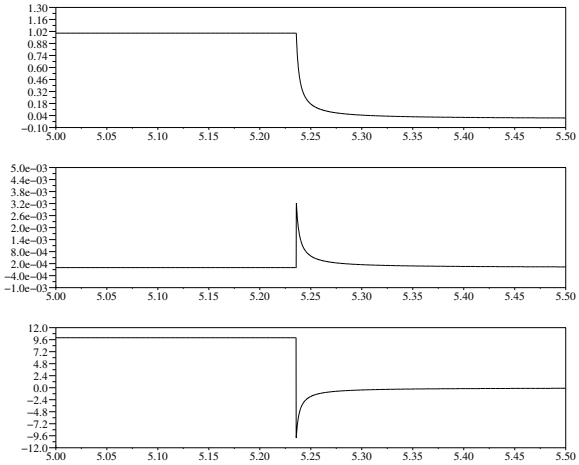

Figure 11: Simulation result for the diagram in Figure 4

cal Mathematics 1 (1994), 265–278.

[4] CAMPBELL, S. L., CHANCELIER, J.-P., AND NIKOUKHAH, R. *Modeling and simulation Scilab/Scicos*, 1st ed. Springer Verlag, 2005.

[5] CAMPBELL, S. L., MOORE, E., NAKASHIGE, R., ZHONG, Y., AND ZOCHLING, R. Constraint preserving integrators for unstructured higher index daes. *Zeitschrift fuer Angewandte Mathematik und Mechanik (ZAMM) 76* (1996), 83–86.

[6] CHANCELIER, J. P., DELEBECQUE, F., GOMEZ, C., GOURSAT, M., NIKOUKHAH, R., AND STEER, S. *An introduction to Scilab*, 1st ed. Springer Verlag, Le Chesnay, France, 2002.

[7] GEAR, C. W. Differential-algebraic equation index transformations. *SIAM. J. Sci. Stat. Comp. 9* (1988), 39–47.

[8] HINDMARSH, A. C. Lsode and lsodi, two new initial value ordinary differential equation solvers. *ACM-Signum Newsletter 15* (1980), 10–11.

[9] NAJAFI, M., AZIL, A., AND NIKOUKHAH, R. Implementation of continuous-time dynamics in scicos. *15th ESS Conference, Delft, the Netherlands* (October 2003).

[10] NAJAFI, M., AZIL, A., AND NIKOUKHAH, R. Extending scicos from system to component level simulation. *ESMC2004 international conference, Paris, France* (October 2004).

[11] NAJAFI, M., AND NIKOUKHAH, R. The use of the numerical integrator in scicos, a user friendly graphical based simulation software. *European Journal of Automation (JESA) Special issue on modeling, formalism, methods and simulation tools* (2006), 95–111.

[12] NAJAFI, M., NIKOUKHAH, R., AND CAMPBELL, S. L. The role of model formulation in dae integration: Experience gained in developing scicos. *17th IMACS World Congress Mathematics and Computers in Simulation, Paris, France* (July 2005).

[13] NAJAFI, M., NIKOUKHAH, R., STEER, S., AND FURIC, S. New features and new challenges in modeling and simulation in scicos. *IEEE conference on control application, Toronto, Canada* (2005).

[14] NIKOUKHAH, R., AND STEER, S. Hybrid systems: modeling and simulation. In *COSY: Math-*

ematical Modelling of Complex System, Lund, Sweden (September 1996).

[15] NIKOUKHAH, R., AND STEER, S. Scicos: A dynamic system builder and simulator, user's guide - version 1.0. Tech. Rep. RT-0207, INRIA Technical Report, Le Chesnay, France, June 1997.

[16] PETZOLD, L. R. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM J. Sci. Stat. Comput 4* (1983).

[17] PETZOLD, L. R. Order results for implicit runge-kutta method applied to differential algebraic systems. *SIAM. J. Numer. Anal. 23* (1986), 837–852.

[18] SHAMPINE, L., REICHELT, M. W., AND KIERZENKA, J. A. Solving index-1 DAEs in MATLAB and Simulink. *j-SIAM-REVIEW 41*, 3 (1999), 538–552.