Selection of Variables in Initialization of Modelica Models

Masoud Najafi

INRIA-Rocquencourt, Domaine de Voluceau, BP 105, 78153, Le Chesnay, France
masoud.najafi@inria.fr

Abstract

In Scicos, a graphical user interface (GUI) has been developed for the initialization of Modelica models. The GUI allows the user to fix/relax variables and parameters of the model as well as change their initial/guess values. The output of the initialization GUI is a pure algebraic system of equations which is solved by a numerical solver. Once the algebraic equations solved, the initial values of the variables are used for the simulation of the Modelica model. When the number of variables of the model is relatively small, the user can identify the variables that can be fixed and can provide the guess values of the variables. But, this task is not straightforward as the number of variables increases. In this paper, we present the way the incidence matrix associated with the equations of the system can be exploited to help the user to select variables to be fixed and to set guess values of the variables during the initialization phase.

Keywords Modelica, initialization, coupling algorithm, numerical solver, Scicos

1. Introduction

Scicos[1] is a free and open source simulation software used for modeling and simulation of hybrid dynamical systems [3, 4]. Scicos is a toolbox of Scilab[2] which is also free and open-source and used for scientific computing. For many applications, the Scilab/Scicos environment provides an open-source alternative to Matlab/Simulink. 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 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 building blocks. They can have several inputs and outputs, continuous-time states, discrete-time states, zero-crossing functions, etc. New custom blocks can be constructed by the user in C and Scilab languages. In order to get an idea of what a simple Scicos hybrid models looks like, a model of a control system has been implemented in Scicos and shown in Figure 1.

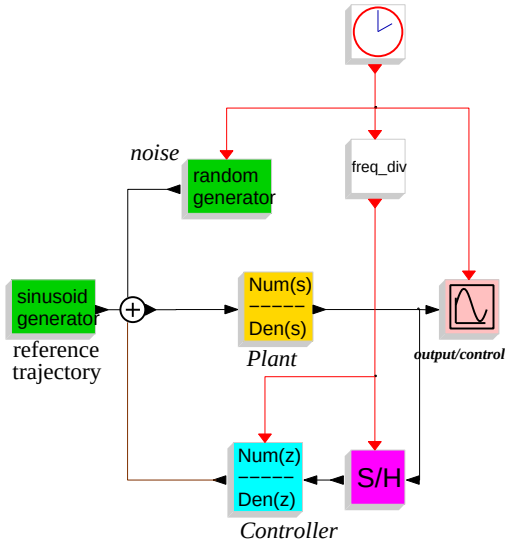

Figure 1. Model of a control system in Scicos

Besides causal or standard blocks, Scicos supports a subset of the Modelica[3] language [7]. The diagram in Figure 2 shows the way a simple DC-DC Buck converter has been modeled in Scicos. The electrical components are modeled with Modelica while the blocks that are used to control the On/Off switch are modeled in standard Scicos.

The Modelica compiler used in Scicos has been developed in the SIMPA[4] project. Recently the ANR[5]/RNTL SIMPA2 project has been launched to develop a more complete Modelica compiler. The main objectives of this project are to extend the Modelica compiler resulted from the SIMPA project to fully support inheritance and hybrid systems, to give the possibility to solve inverse problems

[1] www.scicos.org

[2] www.scilab.org

[3] www.modelica.org

[4] Simulation pour le Procédé et l'Automatique

[5] French National Research Agency

2nd International Workshop on Equation-Based Object-Oriented Languages and Tools. July 8, 2008, Paphos, Cyprus.

EOOLT 2008 website:
http://www.eoolt.org/2008/

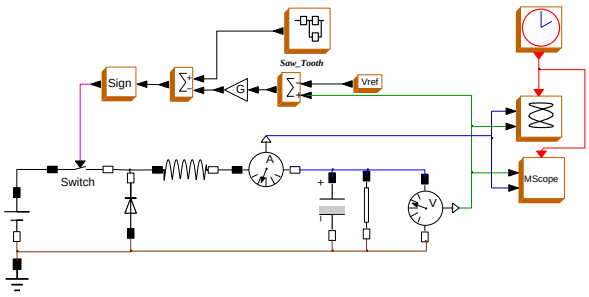

Figure 2. Model of a DC-DC Buck converter in Scicos using Modelica components

by model inversion for static and dynamic systems, and to improve initialization of Modelica models.

An important difficulty when simulating a large Modelica model is the initialization of the model. In fact, a model can be simulated only if it is initialized correctly. The reason lies in the fact that a DAE (Differential-Algebraic Equation) resulting from a Modelica program can be simulated only if the initial values of all the variables as well as their derivatives are known and are consistent.

A DAE associated with a Modelica model can be expressed as

$$0 = F(x', x, y, p) \tag{1}$$

where x, x', y, p are the vector of differential variables of size N_d, derivative of differential variables of size N_d, algebraic variables of size N_a, and model parameters of size N_p, respectively. $F(.)$ is a nonlinear vector function of size $(N_d + N_a)$. Since, the Modelica compiler of Scicos supports index-1 DAEs [1, 2], in this paper we limit ourselves to this class of DAEs.

In Scicos, in order to facilitate the model initialization, the initialization phase and the simulation phase have been separated and two different codes are generated for each phase: The initialization code (an algebraic equation) and the simulation code (a DAE).

In the Initialization phase, x' is considered as an algebraic variable (*i.e.*, dx) and then an algebraic equation is solved. Modelica parameters p are considered as constants unless they are relaxed by the user. There are $(N_d + N_a)$ equations and $(2N_d + N_a + N_p)$ variables and parameters that can be considered as unknowns. In order to have a square problem solvable by the numerical solver, (N_p+N_d) variables/parameters must be fixed. The values of x and p are often fixed and given by the user and the values of dx and y are computed. But the user is free to fix or relax any of variables and parameters. For example, in order to initialize a model at the equilibrium state, dx is fixed and set to zero whereas x is relaxed to be computed. Another example is parameter sizing where the value of a parameter is computed as a function of a fixed variable. In this case, the parameter p is relaxed and the variable x is fixed.

In the simulation phase, the values obtained for x, dx, y, p are used for starting the simulation. During the simulation, the value of p (model parameters) does not change.

In Modelica, the `start` keyword can be used to set the start values of the variables. The start values of derivatives of the variables can be given within the `initial equation` section. For small programs, this method can easily be used but as the program size grows, it becomes difficult to set start values and change the `fixed` attribute of variables or parameters directly in the Modelica program; initialization via modifying the Modelica model is specially difficult for models with multiple levels of inheritance; the visualization and fixing and relaxing of the variables and the parameters are not easy. Furthermore, the user often needs to have a model with several initialization scenarios. For each scenario a copy of the model should be saved.

In Scicos, a GUI has been developed to help the user to initialize the Modelica models. In this GUI, the user can easily change the attributes of the variables and the parameters such as `initial/guess value`, `max`, `min`, `nominal`, etc. Furthermore, it is possible to indicate whether a variable, the derivative of a variable, and a parameter must be fixed or relaxed in the initialization phase.

In the following sections, the initialization methodology for Modelica models and the initialization GUI features will be presented.

2. Initialization and Simulation of Modelica Models

The flowchart in Figure 3 shows how initialization and simulation of Modelica models are carried out in Scicos. The first step in both tasks is removing inheritances. This provides access to all variables and generates a flat model. The flat model is used to generate the initialization and the simulation codes. Note that the initialization data used for starting the simulation is passed to the simulation part by means of an XML file containing all initial values.

In Scicos, three external applications are used in initialization and simulation: `Translator`, `XML2Modelica`, and `ModelicaC`.

`Translator` is used for three purposes:

- Modelica Front-end compiler for the simulation: when called with appropriate options, `Translator` generates a flat Modelica program. For that, `Translator` verifies the syntax and semantics of the Modelica program, applies inheritance rules, generates equations for `connect` expressions, expands `for` loops, handles predefined functions and operators, performs the implicit type conversion, etc. The generated flat model contains all the variables, the derivatives of differential variables, and the parameters defined with attribute `fixed=false`. Constants and parameters with the attribute `fixed=true` are replaced by their numerical values.
- Modelica Front-end for initialization: when called with appropriate options, `Translator` generates a flat Modelica program containing the variables and the parameters defined with attribute `fixed=false`. The derivatives of the variables are replaced by algebraic

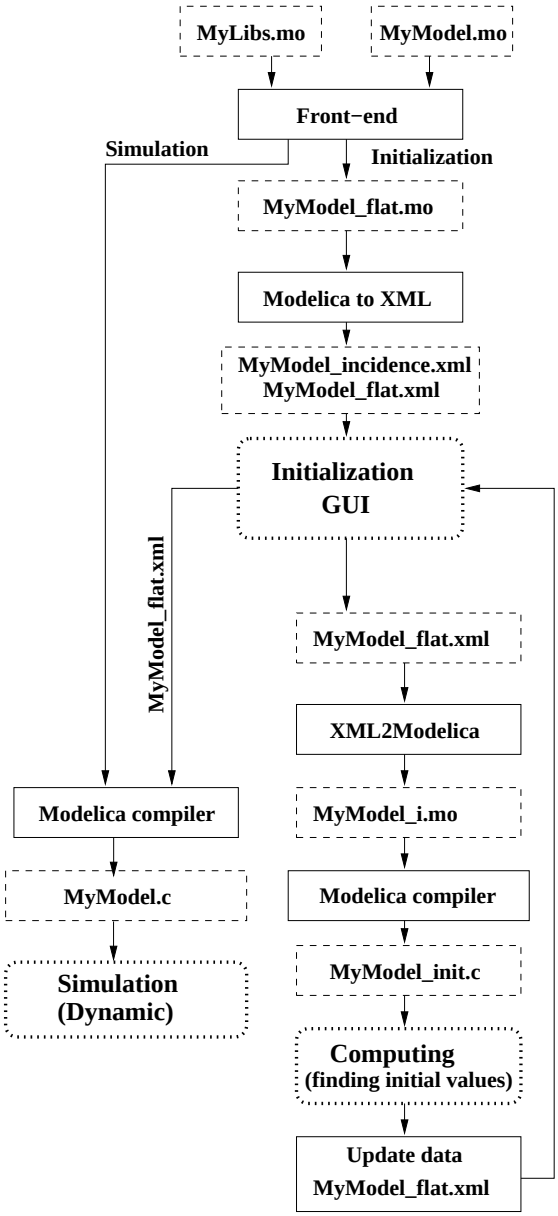

Figure 3. Initialization flowchart in Scicos

variables. Furthermore, the flat code contains the equations defined in the `initial equation` section in the Modelica programs. Constants and parameters with the attribute `fixed=true` are replaced by their numerical values.

- XML generator: when called with -xml option, Translator generates an XML file from a flat Modelica model. The generated XML file contains all the information in the flat model.

Once the XML file generated, the user can change variable and parameter attributes in the XML file with the help of the GUI. The modified XML file have to be reconverted into a Modelica program to be compiled and initialized. This is done by `XML2Modelica`.

`ModelicaC`, which is a compiler for the subset of the Modelica language, compiles a flat Modelica model and generates a C program for the Scicos target. The main features of the compiler are the simplification of the Modelica models and the generation of the C program ready for simulation. It supports zero-crossing and discontinuity handling and provides the analytical Jacobian of the model. It does not support DAEs with index greater than one. Another important feature of the Modelica compiler is the possibility of setting the maximum number of simplification carried out during the code generation phase. Thus, the compiler can be called to generate a C code with no simplification or a C code with as much simplification as possible. This is an important feature for the debugging of the model.

A new feature of `ModelicaC` is generating the incidence matrix. When a C code is generated, the corresponding incidence matrix is generated in an XML file. The incidence matrix is used by the initialization GUI to help the user.

As shown in Figure 3, once the user requests the initialization of the Modelica model, the Modelica front-end generates a flat Modelica model as well as its corresponding XML file. The XML file is then used in the initialization GUI. In the GUI, the user can change the variable and parameter attributes defined in the XML file. The modified XML file is then translated back to a Modelica program. The Modelica program is compiled with the Modelica compiler and a C program is generated. The C program is used by the Scicos simulator to compute the value of unknowns. Once the initialization finished, whether succeeded or failed, the XML file is updated with the most recent results. The GUI automatically reloads and displays the results. The user can then decide whether the simulation can be started or not.

In order to simulate the Modelica model, similar to the model initialization, a flat model is generated. Then, the Modelica compiler simplify the model and generates the simulation code. The generated code is simulated by a numerical solver. The initial values, needed to start the simulation, are read directly from the XML file. The end result of the simulation can also be saved in an another XML file to be used as a starting point for another simulation.

3. Initialization GUI

In Scicos, a GUI can be used for the initialization of the Modelica models. Figure 4 illustrates a screen shot of the GUI corresponding to the Modelica parts of the Scicos diagram of Figure 2. In this GUI, the Modelica model is displayed in the hierarchical from, as shown in Figure 4. Main branches of the tree represent components in the Modelica model. Subbranches are connectors, partial models, etc. If the user clicks on a branch, the variables and parameters defined in that branch are displayed and the user can modify their attributes. In the following subsections, some main features of the GUI will be presented.

3.1 Variable/Parameter Attributes

Any variable/parameter has several attributes which are either imported directly from the Modelica model such as

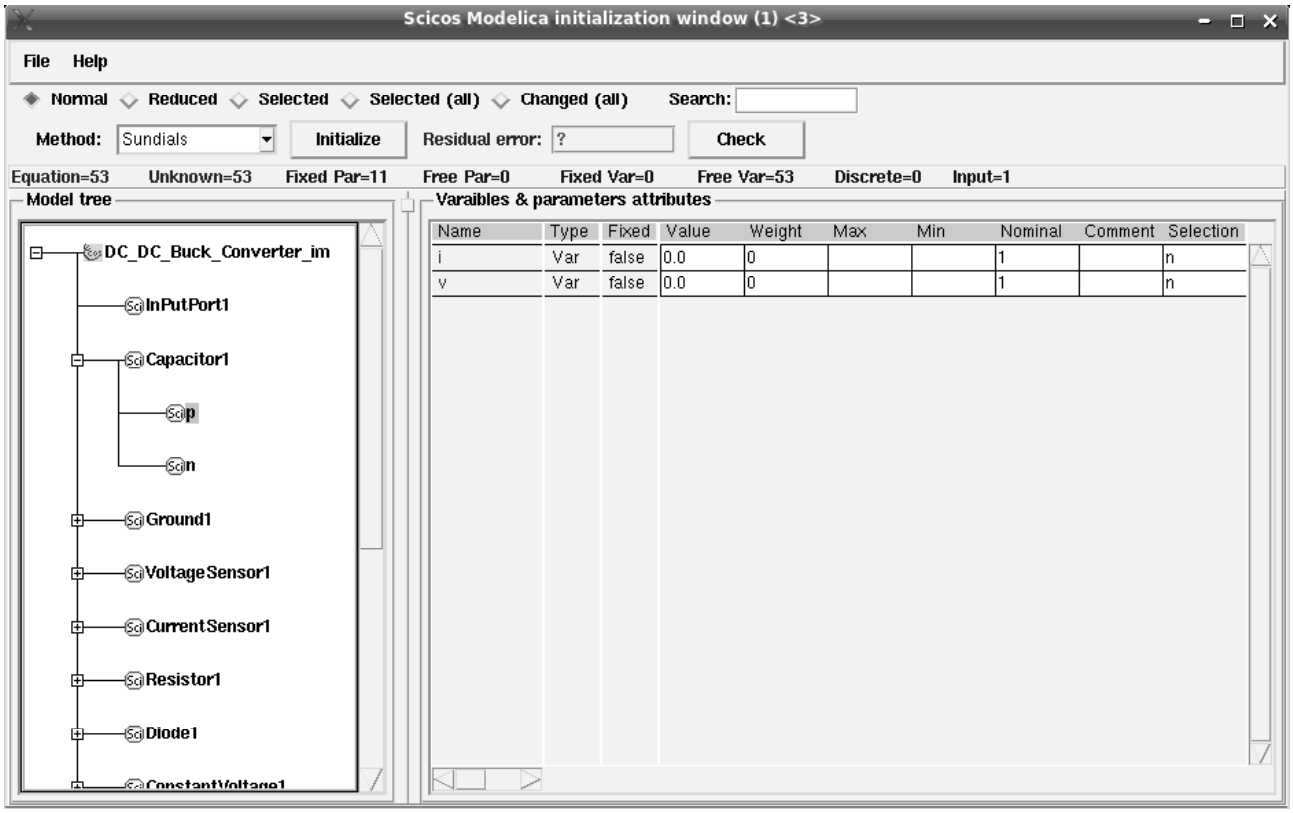

Figure 4. Screen shot of the initialization GUI in Scicos for the electrical circuit of Figure 2

`name`, `type`, `fixed` etc. or defined and used by the GUI *i.e.*, `id` and `selection`.

- `name` is the name of the variable/parameter used in the Modelica program. The user cannot change this attribute in the GUI.
- `id` is an identification of the variable/parameter in the flat Modelica program. The user cannot change this attribute in the GUI.
- `type` indicates whether the original type has been `parameter` or `variable` in the Modelica program. The user cannot change this attribute in the GUI.
- `fixed` represents the value of the `'fixed'` attribute of the variable/parameter in the Modelica program. The user cannot change this attribute in the GUI.
- `weight` is the confidence factor. In the current version of Scicos, it takes either values 0 or 1. `weight=0` corresponds to the `fixed=false` in Modelica whereas `weight=1` corresponds to `fixed=true`. The default value of `weight` for the parameters and differential variables is one, whereas for the algebraic variables and the derivatives of differential variables (converted to variables) is zero.
- `value` is the value of the variable/parameter. If the `weight=1`, the given `value` is considered as the final value and it does not change in the initialization. If `weight=0`, the given `value` is considered as a guess value. If the user does not provide any value, it is automatically set to zero. The user can modify this value in the GUI.
- `selection` is used to mark the variables and parameters. This information will be used by the GUI for selective display of variables/parameters and to influence the Modelica compiler in the model simplification phase.

Note that if the user sets the `weight` attribute of a variable to one, it will be considered as a constant and in the initialization phase it will be replaced by its numerical value. On the other hand, if the user sets the `weight` attribute of a parameter to zero, the parameter will be considered as an unknown and its value will be computed in the initialization phase. This is in particular useful when the user tries to find a parameter value as a function of a variable in the Modelica model.

3.2 Display Modes

Accessing to variables and parameters of the model becomes easier, if different display modes of the GUI are used:

- **Normal** mode is the default display mode. Clicking on each branch of the model tree, the user can visualize/modify the variables/parameters defined in that part of the Modelica model.
- **Reduced** mode is used to display the variables of the simplified model. When the user pushes the initialization button, the flat Modelica model is compiled and a simplified model is generated. In this display mode,

only the remaining variables are displayed. This display mode is in particular useful when the numerical solver cannot converge and the user should help the solver either by influencing the compiler to eliminate the undesirable variables or by giving more accurate guess values.

- **Selected** mode is used to display only the marked variables and parameters of the active branch. A variable or parameter can be marked by putting `'y'` in its selection field in the GUI. By default, all parameters, all differential variables and all algebraic variables whose start values are given are marked. Marking is useful in particular when a branch has many variables/parameters whereas the user is interested in a few ones. In this display mode, unmarked variables/parameters are not shown.
- **Selected (all)** mode is used to display all marked variables and parameters of the Modelica model.
- **Changed** mode is used to display the variables and the parameters whose `weight` attributes have been changed, such as the relaxed parameters.

3.3 Initialization Methods

Once the user modified the attributes of the variables and the parameters, the initialization process can be started by clicking on the "Initialize" button. The initialization consists of calling a numerical solver to solve the final algebraic equation. There are several algebraic solvers available in Scicos such as `Sundials` and `Fsolve` [8, 9, 10].

Once the solver finished the initialization, the obtained results, either successful or not, are put back into the XML file and new values are displayed in the GUI. If the result is not satisfactory, the user can either select another initialization method or help the solver by giving initial values more accurately. This try and error can be continued until satisfactory initialization results are obtained. Then, the simulation can be started.

4. Problems in Variable Fixing and Variable Selection

The initialization of DAE (1) can be formulated as the following algebraic problem

$$0 = F(dx_0, x_0, y_0, p_0) \tag{2}$$

where x_0, dx_0, and y_0 are solutions or the initial values of differential variables, derivative of differential variables, algebraic variables, and parameter values, respectively. The degree of freedom of the equation (2) is $N_d + N_p$, therefore the user should fix $N_d + N_p$ variables or parameters and let the solver find the values of the remaining $N_d + N_a$ unknowns.

Fixing the variables/parameters and giving the start values of the relaxed variables/parameters are essential in the initialization of models. But they are not easy and straightforward for large models. In the next subsections the way these problems are handled in Scicos will be explained.

4.1 Fixing the Variables

Consider the following equation set, composed of two equations and three unknowns.

$$F : \begin{cases} 0 = f(x) \\ 0 = g(x, y, z) \end{cases}$$

Since the degree of freedom is one, the user should provide and fix the value of a variable. But, it is clear that x cannot be fixed, because its value is imposed by the first equation. In this case, the GUI should prevent the user from fixing x.

Consider the next set of equations composed of three equations and five unknowns.

$$F : \begin{cases} 0 = f(x, u) \\ 0 = g(x, z) \\ 0 = h(x, y, z, v) \end{cases} \tag{3}$$

Although the degree of freedom is two, the user cannot fix (u, z), (x, z), or (x, u) at the same time. In general, it is not easy to identify the set of variables that can be fixed. This is in particular important when the number of equations increases. In this case, if the user tries to fix an inadmissible variable, the GUI should raise an error message and prevent the user from fixing the variable.

This problem can be solved using the incidence matrix of the Modelica model. For example, this is the incidence matrix of (3):

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Fixing u and z means removing u and z from the equations which results in the following equation set and the incidence matrix.

$$F : \begin{cases} 0 = f(x, u_0) \\ 0 = g(x, z_0) \\ 0 = h(x, y, z_0, v) \end{cases} \quad \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Although, there are three unknowns and three equations, the incidence matrix is not structurally full rank. This means that u and z cannot be fixed at the same time.

Computing the structurally rank of the incidence matrix is a straightforward way to determine if the user is allowed to fix variables or parameters of the model. Since the incidence matrix is very often large and sparse in practical models, we should use special methods for sparse matrices. In the GUI, a `maximum matching` method (also called a maximum transversal method) is used to compute the structural rank of the incidence matrix. The maximum matching method is a permutation of the matrix so that its k^{th} diagonal is zero-free and $|k|$ is uniquely minimized. With this method, the structural rank of the matrix is the number of non-zero elements of the matrix diagonal [6].

When the user tries to fix a variable or a parameter, the initialization GUI computes the new structural rank of the incidence matrix. If the fixing operation lowers the rank, an error message will raised and the modification will be inhibited.

4.2 Selection of Variables to Be Eliminated

Another recurrent problem in solving algebraic equations is the convergence failure of the solver. Newton methods are convergent if the initial guess values of unknowns are not too far from the solution. So, the user should provide reasonable initial guess values. If the problem size is small and the user knows the nominal values of the unknowns, the user can provide the guess values. But in large models, it is nearly impossible to give all guess values. In medium size Modelica models, we usually end up with models with many variables whose start values are not specified by the user. In this case, their initial guess values are automatically set to zero which is not often a good choice. Furthermore, many variables of a model are redundant and the user does not know for which ones the initial guess should be given. This often happens with variables linked by the `connect` operator in Modelica. Suppose that two Modelica components are connected via a connector, *e.g.*,

```
connect (Block1.x, Block2.y);
```

During the model simplification, the compiler may eliminate either `Block1.x` or `Block1.y`. Even if the user knows the guess values of both, it is not reasonable to ask the user to provide them. Since the user has no influence on the compiler's variable selection, this may cause a problem in solving the initialization equation. Consider, *e.g.*, the following situation.

$$F : \begin{cases} 0 &= \dfrac{x-3}{(x-3)^2+1} - 0.1 \\ 0 &= x - y \end{cases}$$

Here, if the user sets the initial guess of y to 10 and leaves the guess value of x unspecified *i.e.*, $x = 0$, although $y = 10$ is close to the solution, the Newton's method will likely fail. The reason is that the solver ignores the initial value of y and uses that of x. In fact, there is no way to tell the solver the guess value which is "more" correct than the others.

The solution is to formally simplify the equations by eliminating the variables whose guess-values are not given, by replacing them with the variables having given guess-values. For that, in the initialization GUI, variables with known guess-values are marked and the Modelica compiler is told to eliminate the unmarked variables. The user, of course, can modify the list of these marked variables.

The compiler tries to eliminate the variables as much as possible, but a problem may arise when the compiler fails to eliminate all of unmarked variables. Since, the simulator sets their guess-value to zero, the original problem still persists. In this case, the user should be asked to provide the guess-value of the remaining variables. But, usually the user has no idea about the nominal values of the remaining variables or even does not know the physical interpretation of them. As an example, consider the following set of equations for which no guess-values are given.

$$F : \begin{cases} 0 &= f(x) \\ 0 &= x - y \end{cases}$$

Suppose that the compiler eliminates y, but the user does not know the start value of x while y has a physical interpretation and its nominal value can be given. In this case, the initialization GUI should propose to the user all variables that can replace x, *i.e.*, y.

Proposing alternative variables for formal simplification is done in the initialization GUI. In the next sections, it will be shown the way these problems can be handled by the use of the incidence matrix of the model. This is done using the maximum flow algorithms.

5. Maximum Flow Problem

The maximum flow problem is to find the maximum feasible flow through a single-source, single-sink flow network [5]. The maximum flow problem can be seen as a special case of more complex network flow problems. A directed graph or digraph G is an ordered pair $G := (V, A)$ with

- V is the set of vertices or nodes,
- A is the set of ordered pairs of vertices, called directed edges or arcs.

An edge $e = (u, v)$ is considered to be directed from u to v; v is called the head and u is called the tail of the edge; v is said to be a direct successor of u, and u is said to be a direct predecessor of v. The edge (v, u) is called the inverted edge of (u, v).

Given a directed graph $G(V, E)$, where each edge u, v has a capacity $c(u, v)$, the maximal flow f from the source s to the sink t should be found. There are many ways of solving this problem, such as linear programming, Ford-Fulkerson algorithm, Dinitz blocking flow algorithm, etc [12, 11].

5.1 Ford-Fulkerson Algorithm

The *Ford-Fulkerson algorithm* computes the maximum flow in a flow network. The name "Ford-Fulkerson" is often also used for the Edmonds-Karp algorithm, which is a specialization of Ford-Fulkerson. The idea behind the algorithm is very simple: as long as there is a path from the source to the sink, with available capacity on all edges in the path, we send flow along one of these paths. Then we find another path, and so on. A path with available capacity is called an augmenting path.

Algorithm: Consider a graph $G(V, E)$, with capacity $c(u, v)$ and flow $f(u, v) = 0$ for the edge from u to v. We want to find the maximum flow from the source s to the sink t. After every step in the algorithm the following is maintained:

- $f(u, v) \leq c(u, v)$. The flow from u to v does not exceed the capacity.
- $f(u, v) = -f(v, u)$. Maintain the net flow between u and v. If in reality a units are going from u to v, and b units from v to u, maintain $f(u, v) = a - b$ and $f(v, u) = b - a$.
- $\sum_v f(u, v) = 0 \Longleftrightarrow f_{in}(u) = f_{out}(u)$ for all nodes u, except s and t. The amount of flow into a node equals the flow out of the node.

This means that the flow through the network is a legal flow after each round of the algorithm. We define the residual network $G_f(V, E_f)$ to be the network with capacity $c_f(u, v) = c(u, v) - f(u, v)$ and no flow. Notice that it is not certain that $E = E_f$, as sending flow on u, v might close u, v (it is saturated), but open a new edge v, u in the residual network.

1. $f(u, v) \leftarrow 0$ for all edges (u, v)
2. While there is a path p from s to t in G_f, such that $c_f(u, v) > 0$ for all edges $(u, v) \in p$:
   (a) Find $c_f(p) = \min\{c_f(u, v)|(u, v) \in p\}$
   (b) For each edge $(u, v) \in p$
      i. $f(u, v) \leftarrow f(u, v) + c_f(p)$
      ii. $f(v, u) \leftarrow f(v, u) - c_f(p)$

The path p can be found with, *e.g.*, a *breadth-first* search or a *depth-first* search in $G_f(V, E_f)$. The former which is called the Edmonds-Karp algorithm has been implemented in Scicos.

By adding the flow augmenting path to the flow already established in the graph, the maximum flow will be reached when no more flow augmenting paths can be found in the graph. When the capacities are integers, the runtime of Ford-Fulkerson is bounded by $O(E * f_{max})$, where E is the number of edges in the graph and f_{max} is the maximum flow in the graph. This is because each augmenting path can be found in $O(E)$ time and increases the flow by an integer amount which is at least 1. The Edmonds-Karp algorithm that has a guaranteed termination and a runtime independent of the maximum flow value runs in $O(VE^2)$ time.

5.2 Problem of Proposition of Alternative Variables

In order to handle this problem, we build the bipartite graph shown in Figure 5. The left-hand side vertices indicate unknowns, and each vertex at the right-hand side indicates an equation. The edges are bidirectional and their capacity is infinite.

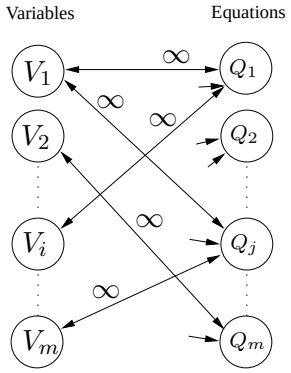

Figure 5. Bipartite graph of variables and equations

Note that, at this stage of initialization, the number of unknowns and the number of equations are identical and the incidence matrix is full rank.

For the problem of proposing alternative variables that can be initialized instead of a variable V_i, based on the bipartite graph in Figure 5, we build another directed graph as shown in Figure 6. In this graph, a source vertex and a target (sink) vertex have been added to the graph. The edge connecting the source vertex to V_i has infinite capacity. All m edges connecting the target vertex to the variable vertices have the capacity 1 (except the edge connected to the vertex V_i). The edges are mono-directional.

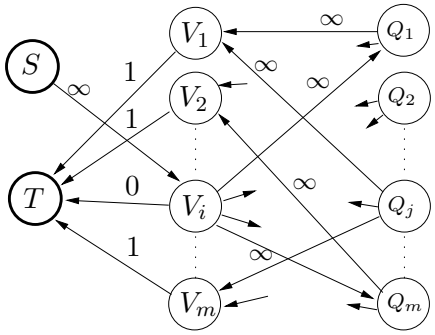

Figure 6. Directed graph for the problem of proposing all alternative variables for V_i

Now, the problem of finding all alternative variables for V_i is transformed into that of finding of all feasible paths from the source to the target. All predecessors of the target are possible alternative variables that can be used instead of V_i. In the initialization GUI, when the user double-clicks on a variable, its alternative variables are displayed. This is a useful help during the initialization.

6. Initialization Iterations

The role of the GUI and the marking in the initialization loop (see the flowchart in the Figure 3) can be summarized in the following algorithm.

1. The GUI automatically marks the model parameters, the differential variables and the algebraic variables whose guess value are given.
2. In the GUI, the user can
   - visualize/modify the `fixed` attribute of the variables and the parameters.
   - change the guess values of variables and parameters (final values if they are fixed).
   - modify whether a variable or a parameter is marked or not.
3. Initialization is invoked.
   - If necessary, the model is compiled. The Modelica compiler tries to reduce the number of unknowns by performing several stages of substituting and elimination. In this phase the marked variables are more likely to be eliminated by the compiler.
   - A numerical solver is used to find the solution of the reduced model.
   - The obtained solution values are send back to the GUI to be displayed.

4. If the obtained results are satisfactory, goto **step 7**.
5. The user can readjust the guess values of the remaining unknowns. If there are still unmarked unknowns in the reduced model, either the user can provide more accurate guess values for them or can click on the variables to see their alternatives variables. The alternative variables should be marked to be remained in the reduced model.
6. Goto step 2
7. Start the simulation

7. Example

The model of a thermo-hydraulic system is shown in Figure 6. In this model, there are a pressure source, two pressure sinks, three pipes (pressure losses), a constant volume chamber, and two flow-meter sensors linked to a Scicos scope.

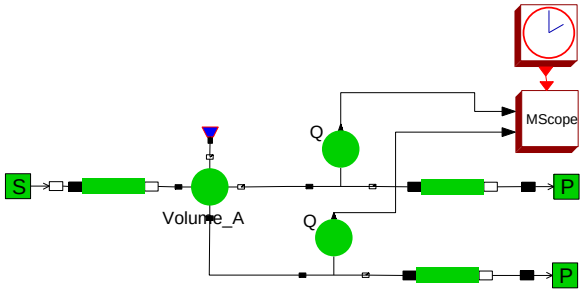

Figure 7. A thermo-hydraulic system

As shown in Figure 8, the initial non-simplified model is composed of 132 equations, 131 relaxed variables and 1 relaxed parameter (*i.e.*, 132 unknowns). The number of fixed parameters and variables are 36 and 1, respectively.

When the model is simplified, the model size is reduced to only 11 unknowns. In Figure 9, where the display mode is `Reduced`, the remaining variables as well as their solution values are shown.

8. Conclusion

In the Modelica models, initialization is an important stage of the simulation. At the initialization, variables and parameters can be fixed or relaxed and their start values can be changed by the user. In this paper, we presented a special GUI to facilitate the task of selecting fixed and relaxed variables.

Acknowledgments

The author would like to thank Sébastien Furic (LMS.Imagine Co.) for a number of helpful comments.
This work is supported by the ANR/SIMPA2-C6E2 project.

References

[1] K. E. Brenan, S. L. Campbell, and L. R. Petzold. Numerical solution of initial-value problems in differential-algebraic equations. *SIAM pubs., Philadelphia*, 1996.

[2] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold. Consistent initial condition calculation for differential-algebraic systems. *SIAM Journal on Scientific Computing*, 19(5):1495–1512, 1998.

[3] S. L. Campbell, J-Ph. Chancelier, and R. Nikoukhah. *Modeling and simulation Scilab/Scicos*. Springer Verlag, 2005.

[4] J. P. Chancelier, F. Delebecque, C. Gomez, M. Goursat, R. Nikoukhah, and S. Steer. *An introduction to Scilab*. Springer Verlag, Le Chesnay, France, 2002.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd edition, 2001.

[6] Timothy A. Davis. *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.

[7] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.

[8] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Softwar 31(3)*, pages 363–396, 2005.

[9] A.C. Hindmarsh. The pvode and ida algorithms. *LLNL technical report UCRL-ID-141558*, 2000.

[10] M. Najafi and R. Nikoukhah. Initialization of modelica models in scicos. *Conference Modelica 2008, Bielefeld, Germany.*, 2008.

[11] Ronald L. Rivest and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill, Inc., New York, NY, USA, 1990.

[12] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 114–122, New York, NY, USA, 1981. ACM.

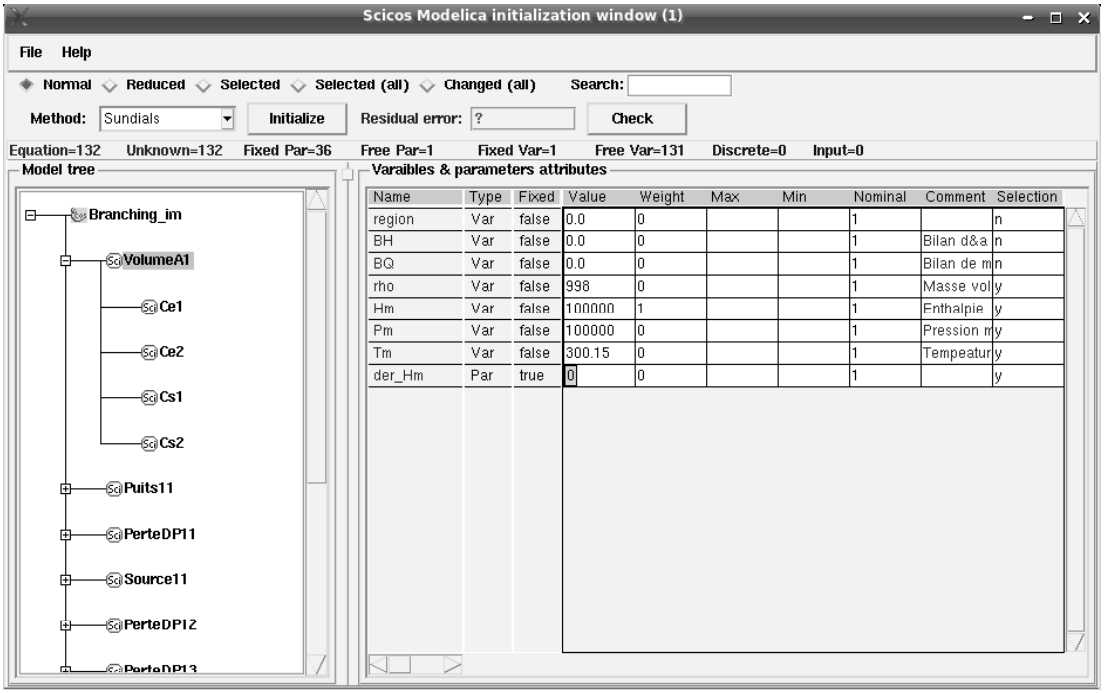

Figure 8. The initialization GUI for the model in Figure 6 (the display mode is `normal` and the variables and the parameters of the block `Volume` are shown)

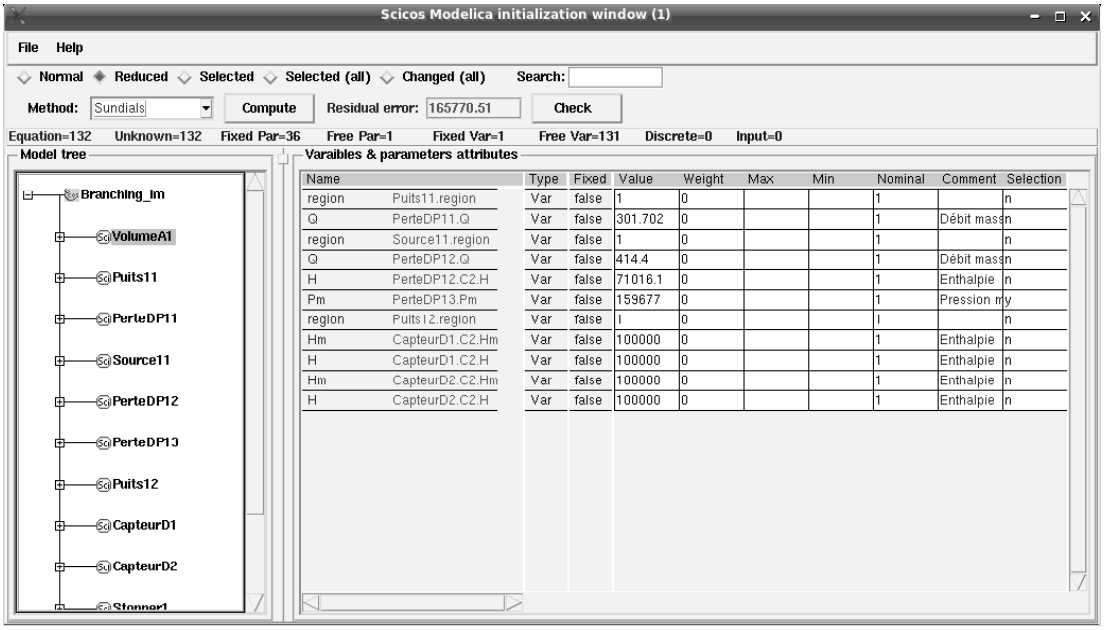

Figure 9. The remaining variables as well as their initial values after the model simplification. The display mode is `Reduced`.