Translating Simulink Models to Modelica using the Nsp Platform

Jean-Philippe Chancelier, Sébastien Furic, Pierre Weis

Translating Simulink Models to Modelica using the Nsp Platform

Jean-Philippe Chancelier[1] Sébastien Furic[2] Pierre Weis[2]

[1]Université Paris-Est, CERMICS (ENPC), F-77455 Marne-la-Vallée cedex 2, France,
jean-philippe.chancelier@enpc.fr
[2]INRIA Paris, 75589 Paris, France, {sebastien.furic,pierre.weis}@inria.fr

Abstract

We present a new Simulink (Simulink) to Modelica (Modelica) translation chain embedded into Nsp. Translated models can be edited (original Simulink diagrams are preserved through translation) and simulated. This translation chain makes use of the Simport tool, originally designed to translate Simulink models to Scicos models, and also relies on Modelicac, i.e. Scicos' Modelica companion compiler.

Using some examples, we demonstrate the effectiveness of the translation process and detail some technical aspects of it. This new Nsp feature extends Nsp's simulation capabilities and makes it a reference platform for users looking for means to simulate Simulink models within a Modelica framework. Resulting Modelica code can even be exported to other Modelica compatible tools.
Keywords: Nsp; Simulink; Modelica

1 Introduction

Nsp is a Matlab-like numerical environment which can run the Scicos modeling environment, a Simulink-like block diagram editor and simulator.

From 2003 to 2008, in the course of funded projects SimPA and SimPA2, a Modelica compiler named Modelicac has been developed allowing Scicos to handle genuine Modelica models. This integration of Modelica within Scicos has been the subject of several papers published at the Modelica conference (Nikoukhah and Najafi, 2008; Nikoukhah and Furic, 2009). The purpose of Modelica is to serve as a high-level description language to extend Scicos expressiveness: Modelica allows users to compose "acausal" models where the original environment forced users to describe their models as block diagrams.

In this paper we focus on a new application of Modelica within Scicos under Nsp, that is as a target language for Simulink model translation.

Several Simulink to Modelica translation tools have already been proposed in the past, we mention in particular Mike Dempsey's Simelica and AdvancedBlocks library (Dempsey, 2003) and Dirk Reusch's Coselica initiative (Reusch). AdvancedBlocks was a fairly complete library of Modelica blocks which allowed users to use Modelica blocks as a one-to-one replacement for Simulink blocks. Up to our knowledge this work remains the most advanced effort in that direction. It is however no longer maintained. Coselica is a library of signal models that allows users to better exploit Modelica from within Scicos by proposing a large set of Modelica submodels in the same spirit as the standard Modelica library (MSL) but with a simpler structure. Many Simulink-like blocks are available in Coselica.

The approach presented here differs from Mike Dempsey's approach in that translation of original blocks is not attempted on a one-to-one basis. Instead, a tool named Simport translates Simulink models by replacing, if necessary, groups of blocks in the original Simulink models with one or several blocks of the target block language (currently, Scicos native of Modelica) so that original semantics is preserved with a high degree of confidence.

We give in this paper a detailed description of this new translation chain hosted by the Nsp environment.

2 Involved Tools

As mentioned above, the translation chain relies on a combination of several tools. We give hereafter a short description of each of them.

2.1 Nsp, a Programming Environment for Numerical Applications

Nsp (Nsp) is a mature Matlab-like Scientific Software Package developed under the GPL license. Nsp features a high-level, safe imperative programming language with automatic memory management. This language can be used interactively, giving users an easy access to efficient numerical routines; It can also be used as a more conventional programming language to extend Nsp's capabilities.

Nsp contains internally a class system with simple inheritance and interface implementation. When used as an interactive computing environment, it comes with online help facilities and an easy access to GUI facilities and graphics.

A large set of libraries are available and it is moreover easy to implement new functionalities. External libraries can also be used: this requires writing some wrapper code (also called *interface*) to live in harmony with Nsp's internal state. The interface mechanism can be either static or dynamic. By using dynamic functionalities one is able to build toolboxes.

Nsp shares many traits with other Matlab-like Scientific Softwares such as Matlab, Octave, ScilabGtk (ScilabGtk;

Campbell et al., 2006), and also with scripting languages such as Python.

The main toolbox used in this work is Scicos that we describe now.

2.2 Scicos, a Block Diagram Modeler and Simulator

Scicos (Scicos) is a graphical dynamical system modeler and simulator originally developed in the Metalau project at INRIA, Paris-Rocquencourt center. With Scicos, users can create block diagrams to model and simulate the dynamics of hybrid dynamical systems and compile models into executable code. Scicos is used for signal processing, systems control, queuing systems, and to study physical and biological systems. Extensions allow generation of component-based modeling of electrical and hydraulic circuits using the Modelica language.

We describe in this paper the Scicos/Nsp version of Scicos maintained and developed at ENPC. Scicos/Nsp is a Nsp toolbox and runs in the Nsp environment. Having access to Nsp functions when designing simulation models is of great importance.

Scicos users often needs to use Nsp functions such as those dedicated to filter design for signal processing or controller design in the construction of simulation models. Nsp programming language can be used for batch processing of multiple simulation tasks, and more generally, models designed by Scicos can be used as functions in Nsp. Nsp graphical facilities can be used for post processing simulation results. But the integration of Scicos and Nsp goes beyond that. Scicos editor is entirely written in Nsp language. This provides many advantages and was in particular of tremendous importance in the current work, indeed: Scicos model data structure is a Nsp structure and thus Scicos models can be programmatically manipulated and build using Nsp scripts. We use this facility in two ways. First to obtain Scicos models from Simulink models, using the fact that the Simport converter produces a Nsp script whose execution in Nsp produces a Scicos model data structure. Second, using Nsp scripts we are able to convert, in a Scicos model data structure, some Scicos blocks to Modelica blocks.

In the conversion process from Simulink to Modelica, the scicos compiler/scheduler also plays a key role. It is used to infer dimensions and types used in the Modelica blocs. This is quite an exciting feature since it gives the possibility to have Modelica blocks for which the Modelica associated model is not a fixed Modelica class but a specific one adapted to specific dimensions and types generated on the fly.

2.3 Modelicac, a Simple Yet Useful Modelica Compiler

Development of Modelicac started in 2003 as a joint work between Inria and TNI-Valiosys (now Dassault Systèmes) in the course of the SimPA (SIMulation pour le Procédé et l'Automatique) french funded project. The goal was to

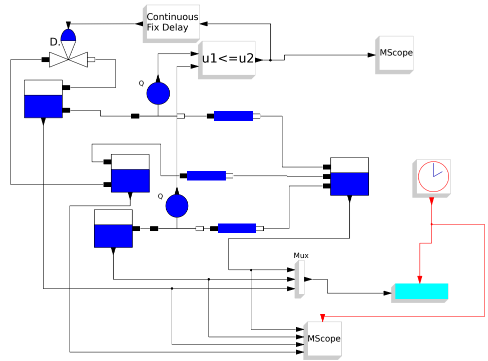

Figure 1. A mixed Scicos-Modelica model as displayed by Scicos's editor

make Scicos compatible with a significant subset of the Modelica language in order for users to be able to describe complex hybrid models without having to resort to low-level block diagram descriptions. Indeed, building a block diagram from a physical model requires 1)performing a *complete* analysis of physical phenomena into play (to determine which elementary blocks to use in the diagram), and 2)determining how data flows between blocks (to connect elementary blocks together). On the other hand, Modelica tools considerably ease physical model construction by automatically analysing the overall structure of physical models described in a much more user-friendly way: familiar physical components (e.g. springs, transistors, hydraulic pumps, etc.) can be used to build models. Translation from this high-level description to low-level data flow is performed automatically in a quite satisfactory way, which frees users from a painful work. Moreover, even slight modifications of physical models may require considerable changes in corresponding block-diagram descriptions; this is not the case with a high-level description.

In its initial version, Modelicac essentially focussed on the "continuous part" of hybrid models. This mainly comprises differential equations and event-trigerring mechanisms (e.g. , "when equations"). Difference equations were however also be described, although with many restrictions, because the idea was to discourage users from writing discrete equations in Modelica. Indeed, Scicos is primarily a hybrid modelling environment and, in particular, it handles discrete, event-trigered changes, much more robustly than Modelica because of its synchronous roots.

In the course of the SimPA project, the Scicos editor has been extended to enable graphical handling of Modelica, native Scicos, as well as hybrid Modelica-Scicos blocks in the same design (see Figure 1).

This combination of synchronous and Modelica-based features offered enough modelling expressiveness to enable useful libraries to be developed. Coselica is one of these libraries, and is one of the ingredients of our Simulink to Modelica translation chain.

In 2005, the funded project SimPA2 started, having as objective the enhancement of the original Modelicac compiler. Among others, support of multiple-file Modelica libraries and interactive initialization of complex hybrid systems have been added.

As a result, the new Modelicac compiler was able to compete with industrial compilers (it even ranked number two in terms of performance on an industrial thermohydraulic benchmark proposed by EDF in 2009).

Today, the Scicos toolbox with its Modelica-compatible extension is freely available under several environments including Scilab, ScicosLab and Nsp.

2.4 Simport, a Simulink Model Importer for Scicos

Simport (Chancelier et al., 2016, 2015) is a comprehensive Simulink import assistant for the Scicos and Altair Activate block system modelers: Simport reads a textual representation of a Simulink model (MDL or SLX file format) and generates the corresponding equivalent Scicos model.

Automating the translation of Simulink models, Simport alleviates the migration process from Simulink to Scicos. Furthermore, Simport allows easy embedding of existing Simulink models or part of models into a Scicos development.

Simport supports a large subset of Simulink basic blocks, but exotic blocks from specific Simulink libraries have no Scicos equivalent; in such a case, Simport generates an empty Scicos super block to incorporate the mandatory hand written Simulink block translation.

Based on compilation techniques, Simport is a fast and reliable translator from Simulink models to Scicos or Altair Activate models.

Simport is distributed with Nsp Scicos (Scicos) and Activate (Altair Activate)

Simport based professional migration services are available by Sciworks Technologies, see `http://sciworkstech.com/`

Simport is a comprehensive Simulink import assistant for Scicos (and Simulate, the Altair's Scicos version)

- Entirely written in the functional language Objective Caml (91 kloc)
- Designed as a compiler (semantics passes + code generation)
- Easy to maintain and extend

Joint work with Jean-Philippe Chancelier, François Delebecque, Clément Franchini, Ramine Nikoukhah, Pierre Weis.

2.4.1 Capabilities

Simport translates Simulink models to Scicos models:

- preserving model hierarchy and diagram topology
- respecting visual aspects of the original model
- aiming to preserve semantics consistency
- supporting both MDL and SLX file formats

2.4.2 Simport translator front-end

From text files, the front-end generates *explicit Simulink model description*

- (1) Parsing (lex/yacc) from MDL text to shadow MDL abstract syntax trees (AST)
- (2) Semantics analysis to obtain deep syntax from shadow syntax (deep syntax contains the semantics of the source)
- (3) Translation from deep syntax to explicit syntax
  - inheritance is made explicit,
  - links encoding is analyzed and translated to Scicos links encoding (generate Scicos split and port blocks)

2.4.3 Simport translator middle-end

From explicit Simulink model description to *abstract target code*

The abstract target code is an abstract API (Application Programmer Interface) for Scicos

- Target code instructions: API function calls for model construction (block instantiation and parameterization, links, ...)
- Middle-end generates abstract target code (lists of API function calls)
- Abstract target code is host language independant (NSP or Scilab for Scicos, HyperMath for Simulate target)

2.4.4 Simulink block translation

From explicit Simulink blocks to corresponding Scicos blocks.

Middle-end maps explicit Simulink model description to *abstract target code*

Middle-end maps Simulink bloks in explicit Simulink models to *abstract target code* that generates Scicos blocks

Middle-end maps Simulink bloks to Scicos blocks using the *block translation library*

2.4.5 Simport *block translation library*

Translation of individual Simulink blocks to abstract target code.

Each Simulink source block is translated either

- into a single basic block, if there exists an equivalent Scicos block, or
- into a *super-block* that implements the Simulink block via a combination of Scicos blocks, or
- into an empty super-block for user completion, if the Simulink block translation is unsupported

2.4.6 Simport *specialized* block translation

Block translation is not a one to one mapping

According to its actual parameters, a Simulink block may

- map to one of two unrelated Scicos basic blocks
- map to a Scicos basic block or a special-purpose Scicos super-block,
- map to a couple of two Scicos super-blocks

2.4.7 Specialized translation for `Abs`

Simulink block `Abs` maps either to a single Scicos basic block, or a super-block, according to sample-time parameter

- if sample-time is inherited (i.e. sample-time parameter is `-1`) then `Abs` maps to the `ABS_VALUEi` Scicos basic block
- otherwise, `Abs` maps to a super-block with a `SampleCLK` Scicos basic block to activate a `ABS_VALUEi` Scicos basic block

Simport translation library covers a large subset of Simulink basic blocks, in particular the so-called *action blocks*

2.4.8 Action block translation

Middle-end maps Simulink action blocks to a pair of super-blocks

- the *actioned block*, a Scicos super-block that implements the Simulink action block body behavior
- the *action event generator*, a Scicos super-block that activates the actioned block according to the Simulink activation specification

Note: Simulink action blocks can observe events (e.g. number of activation so far)

The action event generator outputs data to the actioned block to implement this behavior

2.4.9 Simport back-end: API to host code

The back-end translates abstract target code to concrete code of the Scicos host language

In addition, the back-end provides

- Definition of simulation parameters
- Handling syntax and semantics peculiarities of the host language (e.g. constant π)
- Embedding of various outputs to the host language (e.g. Matlab expressions and Matlab supporting M-files)

2.4.10 SLX file format translation

SLX files are the current output format for Simulink; file format is OPC (Open Packaging Convention): a zip archive with XML files describing the model.

SLX files are translated into MDL shadow abstract syntax trees:

- Uncompress the SLX file
- Parse the model XML files
- Analyze the XML AST
- Translate XML AST to MDL shadow AST

Call the Simport front-end at semantics analysis step and proceed as before

2.4.11 SLX specific difficulties

- Finding MDL equivalent in the verbose XML AST
- Handling new features in SLX that did not exist in MDL

$\rightarrow$ we extend the Simport MDL fragment accordingly

2.4.12 Demo (1)

Given the Simulink model described in Figure 2 and saved as file `model.mdl`.

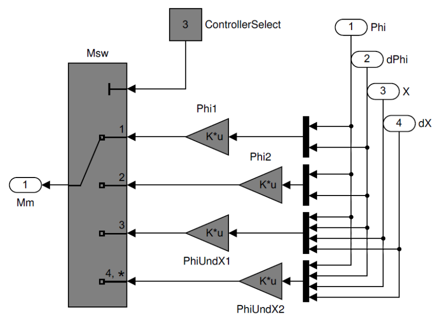

Figure 2. Segway controller as a Simulink model

We translate it into Nsp using the command `simport -tl nsp model.mdl`. We now get file `model.nsp` whose execution in Nsp produce build the Scicos model displayed in Figure 3.

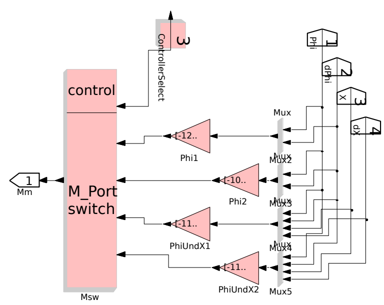

Figure 3. Segway controller as a Scicos model

2.4.13 Demo (2)

Simulation of the Simulink model, gives:

(picture of the Simulink model simulation result)

Simulation of the translated Scicos model gives:

(picture of the Scicos model simulation result)

2.4.14 Limitations

- Some Simulink basic blocks are not covered
- Partial coverage of some semantic constructions (for instance EnableAndTrigger action port)
- No translation for Stateflow and Simscape (no Scicos equivalent)
- No S-function support
- Limited support for Matlab blocks (syntactic translation)

3 Translation from Simulink to Modelica

The Translation from Simulink to Modelica is implemented as a two step process. First, as already described, using Simport, we can translate a Simulink model into a Scicos model. Second, using Nsp scripts we are able to convert a Scicos model into an hybrid Modelica-Scicos model. Conversion is obtained by 1) translation of Scicos blocks to Modelica blocks, and 2) addition in the model of converters along the links which connect Scicos Blocks to Modelica-Scicos blocks. The hybrid Modelica-Scicos models can be edited and simulated in Scicos editor; thus, even if during the Translation process we cannot obtain a full Modelica model[1], the resulting hybrid model may still be used for simulation because users have the possibility to complete untranslated parts thanks to the Scicos editor.

In Figure 4 a Scicos model used to simulate the Lorenz dynamical system is shown. The same model after conversion to Modelica is shown in Figure 5. As it can be seen in Figure 5 the Scopes are not translated to Modelica blocks and converters from Modelica signals to Scicos signals are inserted in the links connected to the entry ports of scopes.

When converters are available, Scicos blocks are replaced by Modelica blocks as a one-to-one process. We have developed a specific library of Modelica blocks to ease the replacement. For example Scicos integrator blocks are replaced by `MB_Integral` Modelica blocks. For some translation we could rely on the already available library Coselica (Reusch), but for many blocks it cannot work because of sizes limitations of Coselica blocks. For example the Coselica `Integrator` is limited to 1-dimensional signals while the Scicos `INTEGRAL_m` block may have n-dimensional entries. One way to encompass that difficulty is to rely on the possibility to generate super blocks for enabling n-dimensional block operations from 1-dimensional basic blocks (See Figure 7 for an example with adder). We have chosen this approach for the converter blocks (See Figure 6) as explained below, but we have also implemented specific blocks which are

[1] In case some blocks are unknown to Simport. Indeed, Simulink blocks are black boxes, so Simport cannot translate blocks or combinations of blocks that are not already described in its tranlation tables.

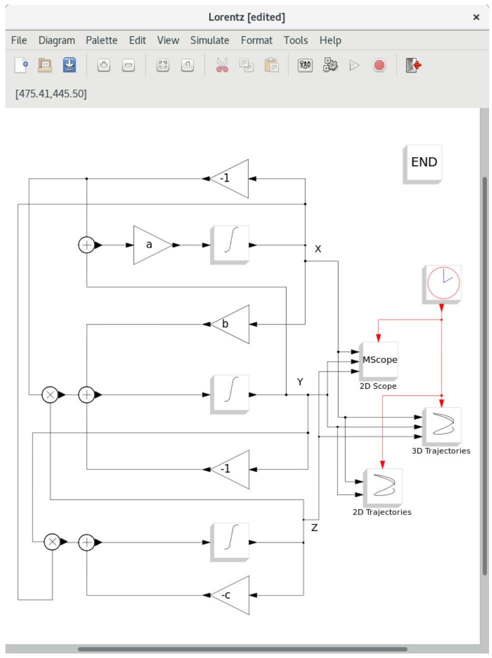

Figure 4. A Scicos model as displayed by Scicos's editor

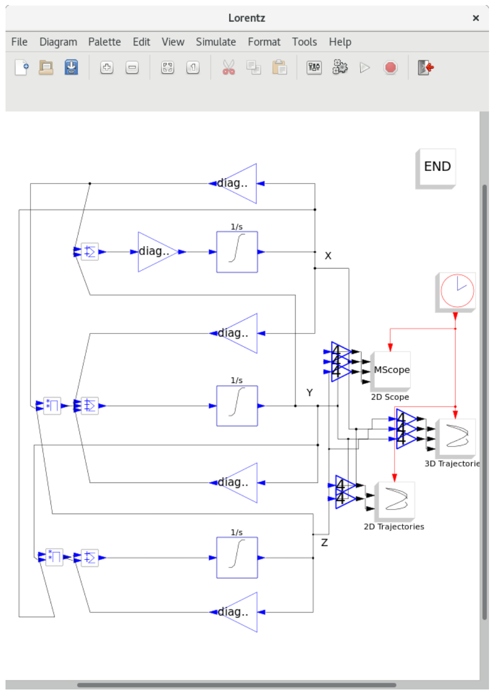

Figure 5. Scicos model after Modelica conversion as displayed by Scicos's editor

able to deal with n-dimensional signals. For example, the `MB_Integral` block is a Modelica-Scicos block which has the particularity that the block at compilation time will dynamically produce a Modelica model for each instance of the block in a specific model. As an example, in Figure 5 each `MB_Integral` Modelica block integrate a 4-dimensional variable without saturation and thus the generated code will be given by

```
model integral2
 parameter Real xinit[   4,1 ] = {{   20 },{   19.9900 },{
      20.0100 },{   20.0110 }};
 RealInput u[4];
 RealOutput y[4](signal(start=xinit[:,1]));
equation
  der(y[1].signal) = u[1].signal;
  der(y[2].signal) = u[2].signal;
  der(y[3].signal) = u[3].signal;
  der(y[4].signal) = u[4].signal;
end integral2;
```

Most of the one-to-one block conversion follows the same mechanism. Building a library of Modelica-Scicos blocks is an on-going work and it only contains around 20 blocks at present. Indeed, this Library can also be used to directly build models in the Scicos editor, it complements the set of Modelica block available in scicos giving access to modelica counterpart of known Scicos blocks.

The one-to-one block conversion is in fact also a multi-step process. We proceed as follows.

First block-to-block conversions are performed but converted Modelica-Scicos blocs are not fully usable because they lack local information (for example the final matrix sizes are unknown at first step). Notice that this first step requires Nsp evaluation of block parameters since they may be used to infer types and dimensions. For example the sizes of a Gain block parameter gives the input/output port sizes of the block, except when the parameter size if 1. But in order to obtain the sizes of a given Gain block parameter we need to evaluate Nsp expressions, since parameters can be given through context (produced by simport from Matlab companion files).

In a second step, links are modified and converters are inserted where appropriate. Notice however that converters sizes are also unknown.

In a third step, sizes and types are obtained by calling the scicos model compiler. However, since the scicos model compiler only infers types and dimension for Scicos blocks this step requires a hidden conversion of the hybrid Modelica-Scicos model into a pure scicos model before trying to infer sizes and types. When sizes and types are inferred for a Modelica-Scicos block, its internal Modelica code can be generated. The code is thus consistent with respect to sizes, types and parameters.

The fact that models can be manipulated and generated programmatically is also used in the conversion process. We illustrate this point by describing more precisely `MB_MO2Sn` the block used to convert Modelica signals to Scicos Signals. The communication between Scicos and Modelica can only be realized using scalar links (for historical reasons, not because of limitations of any of the languages), thus to be able to have converters on

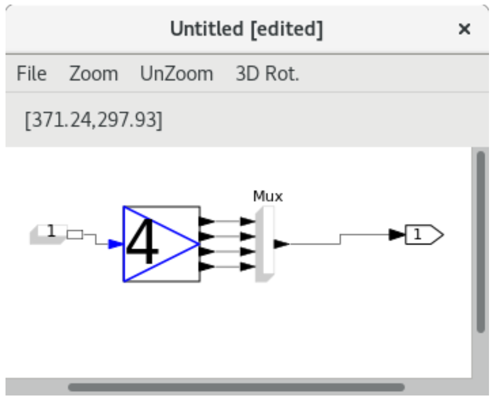

Figure 6. Scicos internal model of a 4-dimensional Modelica to Scicos converter

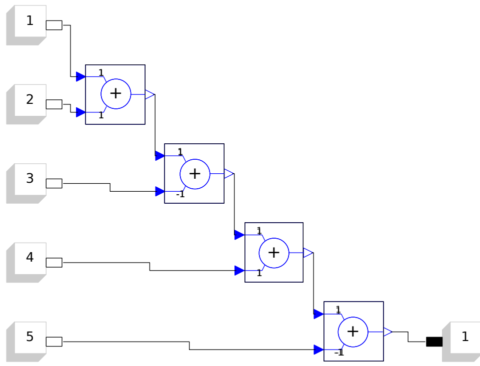

Figure 7. Scicos internal model of a generated 5-dimensional addition block

links which transfer n-dimensional signals we have implemented a block named `MB_MO2Sn` as a super-block. That is, the `MB_MO2Sn` block contains a model and this model is generated dynamically when the used link signal size is known. We give in Figure 6 the internal model of a 4-dimensional Modelica to Scicos converter as used in the model displayed in Figure 5. It contains four 1-dimensional Modelica to Scicos converter. As an other example, to illustrate the possibility to generate models by program we give in Figure 7 an example of a model which performs a 5-dimensional addition of Modelica signals. Implementing a n-dimensional adder block could be implemented that way even if we have chosen to directly embed the Modelica n-dimensional adder block in a unique block.

As already pointed above, during the conversion from Scicos models to Modelica-Scicos models, inferring types and sizes is of utmost importance and it partially relies on Nsp block parameter and context expression evaluation. This is mostly why the conversion cannot completely be performed by Simport. Indeed, inferring types and sizes could be implemented directly in Simport if evaluation of Matlab expression was not required in the process.

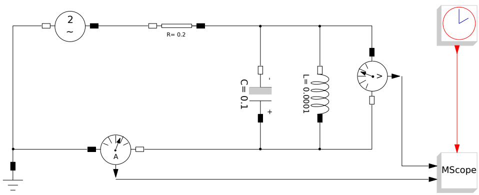

Figure 8. A mixed Scicos-Modelica model of a RLC circuit

3.1 Translation from Modelica to C

Modelica source code is translated to C thanks to the Modelicac compiler. The idea is as follow. Once the model is being run by the user, Scicos gathers all the blocks whose execution semantics is described by means of Modelica code into a unique Modelica program whose source code is given to Modelicac. This program is actually a high-level description of the Modelica part of the model. The following listing illustrates what such a description looks like. It contains the Modelica description generated by Scicos for the model of Figure 8:

```
model RLC_circuit_test_im
  parameter Real VA_VsourceAC_(fixed=false) =
    2.000000e+00  "VA_VsourceAC_";
  parameter Real f_VsourceAC_(fixed=false) =
    1.000000e+00  "f_VsourceAC_";
  parameter Real R_Resistor_(fixed=false) =
    2.000000e-01  "R_Resistor_";
  parameter Real C_Capacitor_(fixed=false) =
    1.000000e-01  "C_Capacitor_";
  parameter Real v_Capacitor_(fixed=false) =
    0.000000e+00  "v_Capacitor_";
  parameter Real L_Inductor_(fixed=false) =
    1.000000e-04  "L_Inductor_";
  VsourceAC      VsourceAC_(VA=VA_VsourceAC_,
                     f=f_VsourceAC_);
  Resistor       Resistor_(R=R_Resistor_);
  Capacitor      Capacitor_(C=C_Capacitor_,
                     v(start=v_Capacitor_));
  Inductor       Inductor_(L=L_Inductor_);
  CurrentSensor  CurrentSensor_;
  Ground         Ground_;
  VoltageSensor  VoltageSensor_;
  OutPutPort     OutPutPort_;
  OutPutPort     OutPutPort_1;
equation
  connect (CurrentSensor_.n,VoltageSensor_.n);
  connect (Capacitor_.p,VoltageSensor_.n);
  connect (Inductor_.p,VoltageSensor_.n);
  connect (Ground_.p,VsourceAC_.n);
  connect (CurrentSensor_.p,VsourceAC_.n);
  connect (VoltageSensor_.p,Resistor_.p);
  connect (Inductor_.n,Resistor_.p);
  connect (Capacitor_.n,Resistor_.p);
  connect (Resistor_.n,VsourceAC_.p);
  CurrentSensor_.i = OutPutPort_.vi;
  VoltageSensor_.v = OutPutPort_1.vi;
end RLC_circuit_test_im;
```

Modelica programs generated by Scicos contain five (possibly empty) sections declaring respectively:

- the parameters of the model,
- the components appearing in the model (i.e. modelica “blocks” used to build the model),
- the connectors to and from the Scicos world (declared as `InPutPorts` and `OutPutPorts`),
- the connection equations (corresponding to links between components of the model, introduced by means of the `connect` keyword), and
- the correspondence between some Scicos ports and some Modelica connectors used to exchange information between both worlds (introduced by means of an equal sign).

From such Modelica programs Modelicac generates native, C-based Scicos blocks. It starts by resolving the names appearing in the Modelica description and instantiates required classes (found in libraries) to form the set of all equations governing the dynamics of the Modelica part of the model. It then flattens the structure of the Modelica model, simplifies equations, and generates C code. The following listing is the result of calling Modelicac with previous Modelica code:

```
/* Scicos block's entry point */

void RLC_circuit_test_im(
  scicos_block *block,
  int flag)
{
  int    *ipar = GetIparPtrs(block);
  double *rpar = GetRparPtrs(block);
  double *z    = GetDstate( block);
  double *x    = GetState(block);
  double *xd   = GetDerState(block);
  double *res  = GetResState(block);
  double **y   = GetOutPtrs(block);
  double **u   = GetInPtrs(block);
  double **work= GetPtrWorkPtrs(block);
  double *g    = GetGPtrs(block);
  double *alpha= NULL;
  double *beta = NULL;
  int *jroot   = GetJrootPtrs(block);
  int *mode    = GetModePtrs( block);
  int nevprt   = GetNevIn( block);
  int *xprop   = GetXpropPtrs(block);

  /* Intermediate variables */
  double v0;

  if (flag == 0) {
    res[0] =
      (x[1]+x[0]*
       (*GetRealOparPtrs(block,3))+
       sin(6.28318530718*
           GetScicosTime(block)*
           (*GetRealOparPtrs(block,2)))*
       (*GetRealOparPtrs(block,1)))*(1.0);
    res[1] = x[2]+ xd[1]*(*GetRealOparPtrs(block,4))-x[0];
    res[2] = xd[2]*(*GetRealOparPtrs(block,6))-x[1];
  } else if (flag == 1) {
    if (!areModesFixed(block)) {
      y[0][0] = x[0]; /* OutPutPort_.vo */
      y[1][0] = -x[1]; /* OutPutPort_1.vo */
    } else {
      y[0][0] = x[0]; /* OutPutPort_.vo */
      y[1][0] = -x[1]; /* OutPutPort_1.vo */
    }
  } else if (flag == 2 && nevprt < 0) {
  } else if (flag == 4) {
    x[0] = 0.0; /* Resistor_.i */
    x[1] = (*GetRealOparPtrs(block,5)); /* Capacitor_.v */
    x[2] = 0.0; /* Inductor_.i */
    if (GetNopar(block)<6){
      SetBlockError(block,-21);
      return;
    }
    SetAjac(block,1);
  } else if (flag == 5) {
  } else if (flag == 6) {
  } else if (flag == 7) {
    xprop[0] = -1; /* Resistor_.i (algebraic) */
    xprop[1] = 1; /* Capacitor_.v (state) */
    xprop[2] = 1; /* Inductor_.i (state) */
  } else if (flag == 9) {
  } else if (flag == 10) {
    alpha=GetAlphaPt(block);
    beta =GetBetaPt(block);
    res[0] = (*GetRealOparPtrs(block,3))*(1.0)*alpha[0];
    res[1] = -alpha[0];
    res[3] = (1.0)*alpha[1];
    res[4] = (*GetRealOparPtrs(block,4))*beta[1];
    v0 = -alpha[1];
    res[5] = v0;
    res[7] = alpha[2];
    res[8] = (*GetRealOparPtrs(block,6))*beta[2];
    res[9] = alpha[0];
    res[12] = v0;
```

```
    }

    return;
}
```

Finally, a new native Scicos block running the generated C code is created by Scicos and connected to the Scicos part of the original model in place of the Modelica blocks. Scicos then performs the simulation of the resulting model.

4 Conclusion and Future Work

In this paper we have shown that NSP can be used as a powerful environment to translate, possibly edit, and simulate some Simulink models. The tool chain comprises two external tools, namely Simport and Modelicac, and the Scicos toolbox with its hybrid Scicos-Modelica library Coselica.

This addition to NSP allows users to simulate many Simulink models with few changes, if any. Moreover, original models can be edited after translation, and possibly connected to native Scicos models, or to modelica models. This opens many interesting perpectives for users willing to run heterogeneous models at a very high level, using the graphical editor provided by Scicos as the sole GUI.

Several enhancements can be made to this preliminary work. The most significant enhancement would probably consists in enriching the translation tables of Simport, to allow more Simulink models to be translated automatically.

5 Acknowledgements

We would like to thank Ramine Nikoukhah from Altair for his considerable help in the design and implementation of the tools we used, in particular Scicos of course, but also Modelicac and Simport.

References

Altair Activate. Multi-Disciplinary System Simulation. URL `https://solidthinking.com/product/activate`.

Stephen Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. Springer, 2006. ISBN: 978-0-387-27802-5.

Jean-Philippe Chancelier, Franąois Delebecque, Clément Franchini, Ramine Nikoukhah, and Pierre Weis. Simport: A Simulink Model Importer for Scicos. In *The 3rd International Workshop on Simulation at the System Level*, École Normale Supérieure de Cachan, France, 2015.

Jean-Philippe Chancelier, Franąois Delebecque, Clément Franchini, Ramine Nikoukhah, and Pierre Weis. Simport. In *ISC'2016 Bucharest*, 2016.

Mike Dempsey. Automatic translation of simulink models into modelica using simelica and the advancedblocks library. In *Proceedings of the 3rd International Modelica Conference, Linköping, Sweden*, pages 115–124, 2003.

Modelica. The modelica language specification. URL `https://modelica.org/documents`.

Ramine Nikoukhah and Sébastien Furic. Towards a full integration of modelica models in the scicos environment. In *Proceedings of the 7th International Modelica Conference, Como, Italy*, pages 641–645, 2009.

Ramine Nikoukhah and Masoud Najafi. Initialization of modelica models in scicos. In *Proceedings of the 6th International Modelica Conference, Bielefeld, Germany*, pages 37–46, 2008.

Nsp. A Numerical computing environment (GPL). URL `http://cermics.enpc.fr/~jpc/nsp-tiddly/mine.html`.

Dirk Reusch. Coselica toolbox für scicoslab. URL `http://www.kybdr.de/software#coselica_toolbox_fuer_scicoslab`.

Scicos. Block diagram modeler/simulator. URL `http://www.scicos.org/`.

ScilabGtk. Gtk+ version of Scilab. URL `http://www.scilabgtk.org`.

Simulink. System modeling and simulation. URL `https://www.mathworks.com/products/simulink.html`.