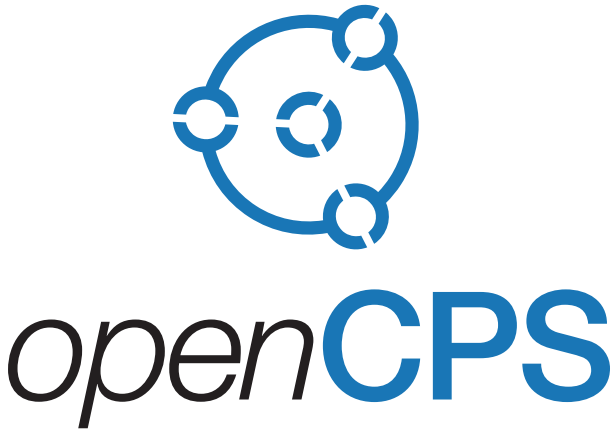
D5.8	Simulink to Modelica importer
Access[1]:	**PU**
Type[2]:	**Report**
Version:	**2.3**
Due Dates[3]:	**M24**



Open Cyber-Physical System Model-Driven Certified Development

Executive summary[4]:

The use of block diagram models is pervasive in industrial modeling, and Simulink models are ubiquitous in the domain. This is a major barrier to the wide adoption of Modelica in industry, since rewriting existing Simulink models into Modelica is a difficult and too much time consuming task. With the *Simulink/Modelica* translator, Inria offers the automatic rewriting of most Simulink models into Modelica. The *Simulink/Modelica* open source translator will greatly facilitate the migration of the industry to open Modelica standards.

[1] Access classification as per definitions in PCA; PU = Public, CO = Confidential. Access classification per deliverable stated in FPP.
[2] Deliverable type according to FPP, note that all non-report deliverables must be accompanied by a deliverable report.
[3] Due month(s) according to FPP.
[4] It is mandatory to provide an executive summary for each deliverable.

Deliverable Contributors:

	Name	Organisation	Primary role in project	Main Author(s)[5]
Deliverable Leader[6]	Pierre Weis	Inria	T5.8 leader	X
Contributing Author(s)[7]	Habib Jreige	Sciworks Technologies	T5.8 member	
	Sébastien Furic	Inria	T5.8 member	
Internal Reviewer(s)[8]	Adrian Pop	Linköpings universitet	WP5 leader	

Document History:

Version	Date	Reason for change	Status[9]
0.1	30/12/2016	First Draft Version	Draft
1.0	08/01/2017	Alpha Version	Released
1.1	28/02/2017	First Issue	Released
2.0	13/11/2017	Alpha version for M24	Draft
2.1	15/11/2017	First release	In review
2.2	16/11/2017	Second release	In review
2.3	24/11/2017	Third release	Released

[5] Indicate Main Author(s) with an "X" in this column.
[6] Deliverable leader according to FPP, role definition in PCA.
[7] Person(s) from contributing partners for the deliverable, expected contributing partners stated in FPP.
[8] Typically person(s) with appropriate expertise to assess deliverable structure and quality.
[9] Status = "Draft", "In Review", "Released".

Contents

Abbreviations

List of abbreviations/acronyms used in document:

Abbreviation	Definition
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
M&S	Modeling and Simulation
N/A	Not Applicable
SotA	State of the Art
TBD	To Be Defined

1 Introduction

The use of block diagram models is pervasive in industrial modeling. Rewriting those block diagram models into Modelica is a difficult and tremendously time consuming task. This is a major barrier to the adoption of Modelica in industry.

As part of the OpenCPS project, Inria proposes the translation of a selected subset of Simulink blocks to Modelica thanks to a *Simulink/Modelica* translator.

The three main tasks are:

1. Task 1 - select a relevant subset of Simulink blocks and constructs.
2. Task 2 - write a parser for Simulink files (including the new SLX format).
3. Task 3 - develop a translator from internal representation of Simulink file to Modelica. The *Simulink/Modelica* translator will be tested on small cases provided by industrial partners.

It should be pointed out here that Inria is not developing a complete translator but rather a tool to facilitate model translation.

An "almost complete" translation would be a very complex and difficult task due in particular to:

1. The semantics of Simulink blocks which, when it exists, is ambiguous and largely dependent on Mathworks proprietary solvers.
2. The absence of input/output typing information for Simulink blocks (the MDL and SLX files do not contain any type information).

Furthermore the implementation of a type reconstruction algorithm to identify port types and parameters of Simulink blocks seems mandatory to translate to Modelica which is strongly typed.

To circumvent this problem, two approaches have been investigated:

1. Retrieve the information of types and dimensions computed by Simulink for the simulation of the model; this involves the use of the available Matlab APIs for querying internal Simulink information.
2. Restrict the generality of Simulink blocks to the simplest cases (scalar inputs/outputs).

A third approach, along the lines of Mike Dempsey's work[Dem03], is to design a new Modelica block library featuring a uniform inputs/outputs port profile (matrix of floating point numbers as in Mike Dempsey's *AdvancedBlocks* library[Dem03]).

Other minor difficulties remain:

1. The links between the Simulink blocks are essentially different from the links of the Modelica blocks,
2. Matlab companion scripts file must be translated into Modelica.

2 Difficulties

2.1 Resource Restrictions

The funding for W5.8 has been divided by 4 compared to the initial expected amount. This shortage of resources has dramatic impact to the W5.8 work package scope and achievements.

2.2 Impact over manpower

So the total manpower available for the project has been reduced to 16 man/months. As a consequence, we could only afford for 10 man/months of expert engineer for the entire project duration.

For the time being, there is only 5 man/months left to complete the work until the end of the project. With such a small amount of work ahead, it is impossible to deliver a full-fledged translator at the end of the project.

We also had to restrict travel expenses, up to canceling all missions to foreign countries for lack of financial resources.

2.3 Impact over Deliverables

As a consequence of these budget restrictions, we had to reconsider our initial objectives:

- we had to restrict the subset of supported Simulink blocks (see 6.1.4),
- we had to restrict the type reconstruction algorithm to the sole determination of matrix sizes (see 6.2), and
- due to the lack of a genuine type reconstruction algorithm, we had to restrict the generality of Simulink blocks: we now only consider blocks with `Real` matrix inputs/outputs (see 6.1.2 and 6.2).

In addition, facing with difficulties with the reuse of existing libraries, we decided to design and implement a dedicated Modelica block library featuring matrix inputs/outputs. See 6.1.

3 Initial Strategy for the *Simulink/Modelica* Importer

Several meetings between Inria, EDF and Sciworks Technologies were organized to adopt the best strategy for the development of the importer.

Two scenarios lie ahead:

1. Generate Modelica code directly from the Simulink model.
2. Base the importer on the *Modelica Standard Library*.

Scenario 1 makes the translator independent of any additional library. This would be the best solution but it needs a lot of development and additional resources which widely exceed the assigned budget (original allocation for Inria was divided by four). This solution has been ruled out.

Scenario 2 has been retained and *Simulink/Modelica* will be based on the *Modelica Standard Library*. Note that *Modelica Standard Library* does not contain all the Modelica counterparts of Simulink blocks selected for the importer, see 4. In addition, even if there exists a *Modelica Standard Library* block similar to some Simulink block, the semantics of the two blocks may be actually different in the particular case at hand. In such a case, consideration is given to either:

1. modify the *Modelica Standard Library* block so that its behavior is that of the Simulink block instance,
2. or create an empty block to be completed by the user.

In the absence of a *Modelica Standard Library* block similar to the Simulink block, we propose either:

1. to create a super-block that reproduces the behavior of the Simulink block to be translated
2. or create an empty block to be completed by the user.

4 Task 1 - Selection of Simulink Blocks

EDF test models are control models that do not contain implicit blocks. After various meetings and technical exchanges on the subject, it was thus decided to limit the importer to **explicit** blocks of Simulink.

The list of supported Simulink blocks have been chosen in close collaboration with our partner EDF and contains more than 56 Simulink blocks; it could evolve while translating EDF test models.

List of *Simulink/Modelica* supported Simulink blocks			
Math Operations	**Continuous**	**Discrete**	**Discontinuities**
Add	Integrator	Discrete-Time Integrator	Dead Zone
Divide	Transfer Fcn	Delay	Quantizer
Gain	Transport Delay	Difference	Rate Limiter
Product		Discrete Transfer Fcn	Relay
Math Function exp, log, pow, sinus, ...			Saturation Dynamic
MinMax			
Polynomial			
Sign			

5 Task 2 - The Lexer/Parser

The *Simulink/Modelica* importer will handle Simulink models both in MDL or SLX source file format. The MDL format is a text format, whereas the SLX format is a compressed binary format which complicates considerably the parsing process. Note also that the Simulink models often use Matlab scripts to define model parameters: those scripts must be parsed to be translated to Modelica.

5.1 The MDL file format

The MDL format is a standard text-based format used by Simulink for saving simulation models. Each MDL file includes a list of elements, each containning a list of values.

The lexer and parser developed by Inria for MDL file format Simulink models are entirely written in OCaml[LDF+11]. The lexical analyzer is based on regular expression techniques and the lexer is produced by the ocamllex automaton generator. The parsing engine is based on LA-LR parsing using the YACC framework. The parser is generated using ocamlyacc.

The parsing phase turns Simulink model source file to shadow abstract syntax trees (model AST), ready for further semantic analyses.

5.2 Testing the parser

To test and validate the Simulink model parser, we wrote a set of tools to automate the test process and assess test results with respect to expected results. All test tools are based on Unix shell scripts and make files.

To test the lexer/parser, we collected many test cases: the origin of these models are:

1. models built on purpose for the case studies,
2. models from various universities, research and industrial centers that are freely accessible from the network,
3. models provided by our industrial partners especially EDF.

We believe that the testsuite currently available is fairly representative in number and covers a wide range of activities: energy, aerospace, automobile, industry, etc.

6 Task 3 - Translation to Modelica

In this section we detail the retained strategy to generate Modelica assemblies from internal representations of Simulink files. As explained above, the intend is not to develop a complete translator—for reasons rooted in semantics, this is not possible— however a fairly decent level of translation can possibly be achieved for a number of interesting models with non-trivial blocks.

Several design choices had to be made, driven in particular by the characteristics of the selected companion library of Modelica blocks used by Simport. Indeed, in this WP we don't want to get "flat" models at the end of the translation process. Instead, we require a certain level of similarity at assembly level between the original Simulink model and its Modelica counterpart, hence the use of a companion library to mimic most of the Simulink blocks of interest. The choice of a suitable companion library was actually the Gordian knot of the task. Indeed, we had to satisfy several requirements:

1. availability as open source code,
2. Simulink orientation (in particular, the capability to deal with matrices of signals),
3. compatibility with a recent version of the Modelica language, and
4. compatibility with the OpenModelica dialect of Modelica (which may differ from other tools regarding interpretation of some language constructs).

In the process of developing our translator, we requested help from OpenModelica developers in order to deal with the last item above. We got helpful recommendations that are mentioned in the next sections (see in particular 6.1.2 and 6.1.3).

6.1 Modelicos: The Companion Modelica Library

We considered several possibilities:

1. use the *Modelica Standard Library*[10],
2. use the *ThermoSysPro* library from EDF,
3. use Mike Dempsey's *AdvancedBlocks* library[Dem03] (that is, the companion library of Claytex Services's Simelica product), and
4. write a new, dedicated library.

The *Modelica Standard Library* and *ThermoSysPro* have been quickly ruled out because their blocks differ too much from Simulink library blocks. In particular, the required matrix nature of port signals is not supported.

AdvancedBlocks from Mike Dempsey was much more appealing than the *Modelica Standard Library*, because this library features most (if not all) the required traits expected in the WP (the library has been purposely designed to match these constraints). Moreover, Mike Dempsey kindly accepted to let us evaluate the library and, in case it would have been useful, to release it as open source.

[10] https://modelica.org/libraries

However, *AdvancedBlocks* was not up-to-date with the current version of the Modelica language[11]. We nevertheless gave a try to the library because of its promising features, and because we thought it was worth the effort to attempt to correct possible illegal Modelica constructs.

We finally did not select *AdvancedBlocks* because none of the examples in the library were able to compile in OpenModelica, and the amount of changes required to correct issues was seen as prohibitive.

We then decided to write a dedicated Modelica library (called Modelicos) in order to satisfy the capability to deal with matrices of signals and the compatibility with OpenModelica. We give hereafter a description of the design decisions.

6.1.1 Computational Causality

Modelica is an "acausal" language, meaning that one can let conforming tools decide (typically by means of graph-based algorithms) how to automatically assign computational causalities in models into which relations between signals are described by means of *equations* instead of imperative statements. This approach, although flexible for the model designer, does not guarantee a smooth transition from model design to simulation results. The reason is that most of the time, when several models involving equations are being composed, the resulting system of equations does not map to a unique data flow: indeed, there are several possible ways to decompose model constraints into elementary computations. Issues arise when the dependency graph of the system of equations cannot be simplified into a tree (i.e. the so-called BLT decomposition contains irreducible blocks of size greater than 1). Corresponding non-linear constraints may have zero, one or more than one solution. This is problematic in practice because a numerical solver may not found a solution (even if it is unique) or may find a wrong solution for instance. In order to avoid this, a causal data flow approach is preferable. In this case, a successful model composition means that the data flow is well defined and that chances are better to obtain meaningful simulation results. Consequently, we adopted the causal data flow approach as a design principle in our library.

Blocks in our library only feature explicit input or output ports carrying matrices of signals in and out of the block, respectively. Moreover, blocks involving delays such as integrators can be used to "break" loops in models because in such blocks outputs depend solely on the past of inputs.

6.1.2 Ports and Matrix Sizes

Ports in the Modelica companion library must carry matrices of signals. Simple scalar signals can efficiently be exchanged through ports once they have been lifted as 1×1 matrices.

In our library, real ports have the following definition:

```
within Modelicos.Ports.Input;
```

[11] Communication from Mike Dempsey on July 2017

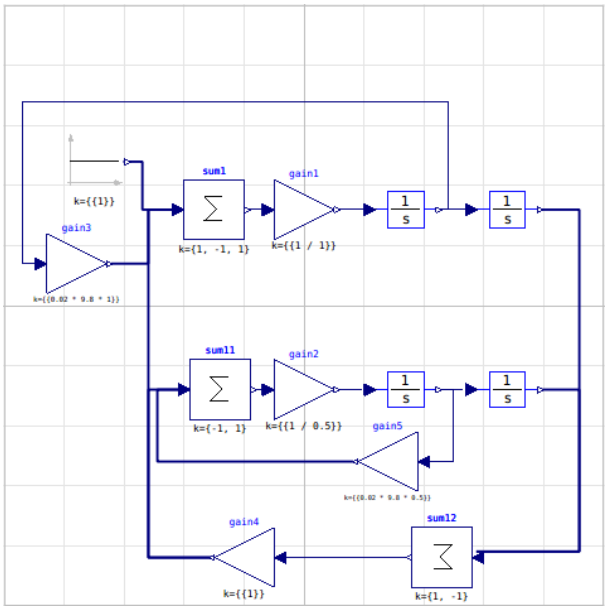


Figure 1: Control example using Modelicos

```
connector RealPort
  parameter Integer m, n;
  input Real s[m, n];
  annotation (...);
end RealPort;
```

```
within Modelicos.Ports.Output;

connector RealPort
  parameter Integer m, n;
  output Real s[m, n];
  annotation (...);
end RealPort;
```

One can notice the parameters `m` and `n` which represent the size of rows and columns, respectively. The values of these parameters were, in the initial design of the library, supposed to be automatically transmitted from block to block through connections, each block being responsible of internal propagation from inputs to outputs as illustrated in the following example:

```
within Modelicos.Blocks.Math;

block Gain "Output the product of a gain value with the input signal"
  import Modelicos.Ports.*;
  parameter Real[:, :] k;
  Input.RealPort u
    "Input signal connector" annotation (...);
  Output.RealPort y(final m = u.m, final n = u.n)
    "Output signal connector" annotation (...);
equation
  if size(k, 1) == 1 and size(k, 2) == 1 then
    y.s = k[1, 1] * u.s;
  else
```

```
    y.s = k .* u.s;
  end if;
  annotation (...);
end Gain;
```

However, a limitation of OpenModelica forced us to change our initial parameter propagation strategy. Indeed, as soon as a model contains a loop, OpenModelica refuses to compile it, complaining that some types are inconsistent in connection equations. As an illustration of this, consider the assembly of 1. In this model, we have two closed loops (which do not correspond to algebraic loops in the final Modelica because we took care of designing the library as to avoid issues with "acausality"). OpenModelica currently does not accept this model. According to discussions with OpenModelica developers, this is because OpenModelica considers individual blocks as atomic, hence it does not see that, within each integrator in the model, loops are broken and parameters can be propagated (there is no algebraic loop).

As a consequence, we decided to adapt the work-flow in Simport to generate final values of parameters although it was not designed to perform such model simplification in the first place (propagation of information from block to block is supposed to be performed by the target language environment).

6.1.3 Imperative Statements in Blocks

Execution semantics of blocks are defined by means of imperative statements (this is possible since all the blocks in Modelicos have explicit input/output ports). Imperative statements are described by means of *algorithms* in Modelica, an imperative sub-language comprising traditional *for* loops, assignments, etc.

The following listing illustrates the use of imperative statements in Modelicos:

```
within Modelicos.Blocks.Math;

block Sum2 "Sum"
  import Modelicos.Ports.*;
  extends .Modelica.Blocks.Icons.Block;
  parameter Integer n(min = 1) = 1 "Number of inputs";
  parameter Real k[n] = ones(n);
  Input.RealPort u[n]
    "Real input ports" annotation (...);
  Output.RealPort y(final m = u[1].m, final n = u[1].n)
    "Real output port" annotation (...);
  annotation (...);
algorithm
  y.s := zeros(y.m, y.n);
  for i in 1:n loop
    y.s := y.s + k[i] * u[i].s;
  end for;
  annotation (...);
end Sum2;
```

Because of issues with imperative statements (wrong numerical results), we had to rewrite some blocks.

Here is a version of the previous block which gives correct numerical results according to our tests:

```
within Modelicos.Blocks.Math;

block Sum "Sum"
  import Modelicos.Ports.*;
  extends .Modelica.Blocks.Icons.Block;
  parameter Integer n(min = 1) = 1 "Number of inputs";
  parameter Real k[n] = ones(n);
  Input.RealPort u[n]
    "Real input ports" annotation (...);
  Output.RealPort y(final m = u[1].m, final n = u[1].n)
    "Real output port" annotation (...);
  annotation (...);
protected
  function f
    input Integer i;
    input Real[:] k;
    input Real[:, :, :] u_s;
    input Real[:, :] acc_in;
    output Real[size(acc_in, 1), size(acc_in, 2)] acc_out;
  algorithm
    if i == 0 then
      acc_out := acc_in;
    else
      acc_out := f(i - 1, k, u_s, acc_in + k[i] * u_s[i]);
    end if;
  end f;
equation
  y.s = f(n, k, u.s, zeros(y.m, y.n));
  annotation (...);
end Sum;
```

We learned from discussions with OpenModelica developers that the wrong results obtained with the initial version were due to a problem with one of the optimization modules of OpenModelica (namely, `removeSimpleEquations`). Fortunately, this can be circumvented by means of the following compiler flag:

```
--preOptModules-=removeSimpleEquations
```

so finally the “nice” version of the `Sum` block also gives us correct results.

6.1.4 Supported Blocks

Modelicos is intended to serve as proof of concept for the feasibility of Simulink to Modelica translation. We have implemented several common blocks frequently used in Simulink models, among which gains, adders, constants and integrators. Users will have the possibility to enrich this library by adding new Modelica block definitions, and by parameterizing Simport accordingly.

6.2 OpenModelica as a New Target of Simport

Simport initially targets simulation environments having strong similarities with Matlab/Simulink (e.g., NSP/Scicos). By designing a new library suitable for translation of many Simulink models, we made OpenModelica a possible new target for Simport through the use of Modelicos as an intermediate layer. However, compared to implementation efforts required for other targets, some additional checks and model processing are necessary for OpenModelica because this environment requires precise type information to be explicitly given with the declaration of ports (Modelica compilers do not *infer* types from syntax of programs, in particular size of matrices, see 6.1.2). *Type reconstruction* could have been used to handle this, however this would have required considerable work which, given the resources allocated to the project, cannot be reasonably implemented. We then decided to make use of one of Simport's targets (namely, NSP) to handle computation of matrix sizes in port declarations: NSP processes the model, generating an intermediate file containing size information which is reread by Simport to generate the final Modelica code.

This required some additional work in the NSP/Scicos system.

In any case, adequacy between the Simulink and the translated model is impossible to prove in general: source and target semantics differ and we have no easy means to statically detect semantic issues between Simulink models and generated Modelica models. However this fundamental problem already occurs with all Simport targets. Indeed, the semantics of Simulink itself is not precisely known (it is given informally in the user documentation of models). As a consequence, this verification is left to the user.

Semantic issues apart, adding a new target to Simport is not a hard task as Simport has precisely been designed to be enriched with new targets (again, provided these targets reasonably mimic Simulink semantics). So, a new target called Modelicos has then been added Simport, whose purpose is twofold:

- generate Modelica code (compatible with OpenModelica) of an *assembly* of Modelicos blocks corresponding to original Simulink models, and
- generate Modelica annotations to allow graphical description of original Simulink models to be preserved in translated models.

6.3 Status of Software Implementation

We experienced some delay due to issues mentioned in previous sections and also because in the first year of the project Inria didn't find a software developer with enough Modelica skills to handle the task. Sébastien Furic joined Inria and started working on Modelica specific aspects of the project since June 2017.

We started by investigating capabilities of available tools to implement Modelica specific features required by the project.

Given this information, five tasks have been identified:

- design and implement the intermediate layer Modelicos,
- generate array size information (by making NSP generating missing information),

- describe and implemente the mapping between Simulink and Modelicos assemblies, and
- describe and implement the mapping between Simulink graphics and Modelica annotations.

The following table summarize the progress:

Sub-task	Completion
Progress of auxiliary sub-tasks	
Determination of constraints and limitations	100%
Design and implementation of Modelicos	20%
Generation of array size information	100%
Progress of Simport specific sub-tasks	
MDL parser	100%
SLX parser	100%
Description and implementation of block assemblies	0%
Description and implementation of graphic layers	0%

References

[Dem03] *Automatic translation of Simulink models into Modelica using Simelica and the AdvancedBlocks library*, volume 3 of *Proceedings of Modelica*, Linköping, Sweden, 2003. The Modelica Association (http://www.Modelica.org) and Programming Environments Laboratory, Institutionen för datavetenskap, Linköpings universitet (http://www.ida.liu.se/ pelab).

[LDF+11] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. *The OCaml system (release 3.12): Documentation and user's manual*. Institut National de Recherche en Informatique et en Automatique, July 2011. URL: http://caml.inria.fr/distrib/ocaml-3.12/ocaml-3.12-refman.pdf.