
Export of Modelica models into ProMoVis

Jesper W. Moberg

Luleå University of Technology

DRAFT , 2012-06-17

ABSTRACT

ProMoVis provides an environment where engineers may, through a combination of graphical visualisation and powerful analysis tools, analyse and deepen their understanding of complex processes and ~~its~~ components. Currently a drawback is that the user has to create a model of the **system**, **through the graphical interface that ProMoVis provides,** **even though he may have existing models created in another environment.**

This **report** describes the design process and implementation of a **tool, written in python,** **that utilizes the JModelica environment to export models, described in Modelica, to an equivalent SFG representation used in ProMoVis.** This is done through linearising a model around an operating point and then extracting a **DAE** representation of it. From this DAE, it is then possible to extract an **SFG** representation, describing how a component depends on other components in the system.

Besides demonstrating the key algorithm for extracting the SFGs this report also discusses the design of the software and how the **internals** of the tool is divided into separate **phases**. Phases responsible for compiling the **original model**, building an intermediate representation of the SFGs for internal use, validation of the system together with some optional transformations of the original model and finally the actual generation of the ProMoVis representation from the intermediate representation.



PREFACE

I worked on this project during the spring of 2012 starting out as a novice in the field of control theory and physical modeling with only the most fundamental knowledge in the subject. But having been in touch with Modelica in earlier courses, and found it to be a powerful tool that **i** would like to deepen my understanding of. It was not hard to accept when **Wolfgang** proposed this project.

The source code to the tool is currently available at <https://github.com/jwmborg/Modelica-to-PromoVis-export>, although i probably will move the actual export software to a separate repository, without the school related information, for continued development after this project has ended.

I would like to thank Wolfgang Birk and Miguel Castaño for the time and effort they have put down, with their pragmatic approach, they have been invaluable in giving me a deepened understanding of the field of control theory and mathematical modeling.

Finally, i would like to give credit to Johan Carlson for providing the L^AT_EX template **upon which this report is based.**

Hopefully the output of the project, the export tool, might help make ProMoVis more accessible to existing users of Modelica and also provide a foundation for a tighter integration between the two environments.

Jesper Moberg

CONTENTS

CHAPTER 1 – INTRODUCTION	1
1.1 Background	1
1.2 Problem Description	1
1.3 Project Goal	1
CHAPTER 2 – METHOD	3
2.1 Workflow	3
2.2 Tools	3
2.2.1 Modelica	3
2.2.2 JModelica	3
2.2.3 Numpy and SciPy	4
2.2.4 FIXME IS THERE A NEED TO Review THE CONCEPTS OF THE SFG	4
CHAPTER 3 – IMPLEMENTATION	5
3.1 General structure	5
3.1.1 JModelica(Compile)Phase	5
3.1.2 Generation of the internal structure	6
3.1.3 Validation and Transformations	7
3.1.4 Scenario generation	8
3.2 Interfacing with the tool	9
3.3 DAE vs ODE	10
CHAPTER 4 – RESULT	11
4.1 A small example	11
4.1.1 The input model	11
4.1.2 Running the export	12
CHAPTER 5 – DISCUSSION	17
5.1 Discussion	17
5.1.1 The project	17
5.1.2 Limitations of the tool	18
5.2 Closing remarks	19
APPENDIX A – EXTRACTING THE SFG’S IN THE CORRECT ORDER	21



A.1	Deciding which row to solve for which variable	21
APPENDIX B – ODE vs DAE REPRESENTATION OF SYSTEMS EXTRACTED FROM MODELICA		25
APPENDIX C – NOTES FROM PROJECT MEETINGS		29
C.1	2012-02-27	29
C.2	2012-04-02	29
C.3	2012-04-18	30
C.4	2012-04-27	31

CHAPTER 1



Introduction



1.1 Background

 While Modelica, Simulink and other existing tools **is** very powerful when it comes to simulation and modelling of complex systems. A drawback with existing tools is that  though a user declares the relations between inputs and outputs, **at simulation time** **models are still assumed to be treated as "black boxes"**. Making it hard to **monitor and analyse the internal behaviour of components in the modelled systems**. Due to an identified need for a tool that can solve these problems [1], the design and implementation of ProMoVis started in 2010 and is currently **being prepared for release** as an open source project.

1.2 Problem Description

As stated earlier, ProMoVis purpose is visualisation and analysis of large systems. It assumes that the user has information about the components of the system and the relations between them. Today, there is no  **scripting interface** to ProMoVis and models have to be created in the graphical environment,  placing the components and defining the relations one by one. This is a task that can become very tedious and error-prone for **complex** systems. At the same time a user that would benefit from using ProMoVis in his design process **can be assumed to already have existing models, from another environment, available**. Therefore the possibility to import existing models would make ProMoVis more accessible to its intended users.

1.3 **Project Goal**

In recent time Modelica, as a tool for modeling of physical systems, has shown an increase in popularity and has become somewhat of a standard for describing physical systems.

Due to its widespread use and the amount of open source tools that are available for models described with help from the language, this projects goal is to implement a tool that can import models, described in Modelica, into ProMoVis.

CHAPTER 2

Method

2.1 Workflow

During the projects initial phase much of the time was put on familiarizing with the ProMoVis and JModelica environments and the math that acts as a foundation for the Modelica models. During these first weeks there were sporadic meetings and discussions with the supervisor whenever problems or questions appeared. After an initial meeting with the developer of the ProMoVis frontend (See Appendix C). The start of the development of a first "Proof of concept" was initiated. After finishing this, a more thorough design phase started where both the developers of ProMoVis and a "test-user" ~~was~~ involved.

2.2 Tools

2.2.1 Modelica

Modelica is [an](#) object-oriented declarative programming language used to create models of physical system. Unlike many other modelling languages, Modelica is not domain specific and thus can be used to model physical systems consisting of a mix of electrical, mechanical and chemical processes. Since the release of the first language specification in 1997 the continuous development of the language has been maintained by the Modelica Association and the current version of the Language specification is 3.2 [3].

2.2.2 JModelica

The initial demand on the environment to be used was that it should be Open-source. After examining the options available JModelica[4] was chosen because of its Python front-end, making it easy to interface with the tool from other software, while still providing a structured environment around which we can build all parts of the export tool.

2.2.3 Numpy and SciPy

Another important feature supporting the use of the JModelica platform was its tight integration with the NumPy and SciPy packages [5]. These are two very popular Open source packages for Python providing a strong and well documented toolbox for mathematical computing. When extracting the mathematical models through JModelica they are represented using the data-structures provided by NumPy and in many cases this removes the need to "reinvent the wheel" for many of the operations that needs to be performed on the mathematical models before export. Instead one can directly use existing methods provided by SciPy and NumPy. The use of [this package](#) also makes the source-code for the export tool more accesible to other programmers familiar with Python and these packages.

2.2.4 **FIXME IS THERE A NEED TO Review THE CONCEPTS OF THE SFG**

And how it is used in promovis.



CHAPTER 3

Implementation

3.1 General structure

The internal structure are divided into four explicit phases.

3.1.1 JModelica(Compile)Phase

The JModelica environment is used to compile the Modelica models to JModelicas JMU representation [4]. ProMoVis assumes the systems to be linear and the compiled model needs to be linearised around an operating point. The linearisation is performed at time 0, and it is assumed that the system is stable at the time.

After linearisation a DAE on the following form can be extracted:

$$E * dx = A * x + B * u + F * w + g \quad (3.1)$$

This representation should be familiar, the x and dx vectors represents the states and outputs of the linearized system. The u vector represents declared inputs. W is algebraic variables, that is all variables in the original model that has no derivative declared. Finally g is a constant bias vector.

The linearisation also outputs some useful information that we later use in the generation of the ProMoVis scenarios:

- *State names*, corresponding to the declared variable names from the original Modelica file.
- *Input names*, corresponding to the declared input names from the original Modelica file.

- *Algebraic names*, corresponding to the declared algebraic variable names from the original Modelica file.
- *Operating points* for the linear model $dx0, w0, u0$ and $x0$. Which is, in later stages, used to provide feedback for the user regarding the validity of the resulting model.

From now on we will refer to states and algebraic variables as simply "variables" and whenever a distinction between the two is needed, it will be explicitly declared.

3.1.2 Generation of the internal structure

When the DAE is obtained, the task is then to extract the relations between the variables from it. The tool collects and stores each of the variables, together with each of the variables that acts as inputs to it. Although it might feel more natural to store a variable together with each of the variables that it affects, this representation was chosen since ProMoVis supports both of the representations but it is straightforward, from the DAE, to solve a variable for all its inputs.

This fact is easily seen with the following example.

The DAE for a model with two state variables, two inputs and two algebraic variables could be described by the following general structure:

$$\begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \\ E_{31} & E_{32} \\ E_{41} & E_{42} \end{bmatrix} \begin{bmatrix} x0 \\ x1 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \\ A_{41} & A_{42} \end{bmatrix} \begin{bmatrix} x0 \\ x1 \end{bmatrix} + \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \\ B_{41} & B_{42} \end{bmatrix} \begin{bmatrix} u0 \\ u1 \end{bmatrix} + \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \\ F_{31} & F_{32} \\ F_{41} & F_{42} \end{bmatrix} \begin{bmatrix} w0 \\ w1 \end{bmatrix}$$

The task is now, to find which of the rows in the system that should be solved for which variable so that the whole system can be solved. Here a demand is that the DAE may not contain any Algebraic loops. For a more thorough discussion regarding this and the decision of which row to solve for which variable please review Appendix A

After deciding which row that should be solved for which of the variables we build up an intermediate representation of the SFG, used internally in the tool, where we store each of the variables together with all of its input variables. This is best explained with the help of an example, if we assume that we should solve row 1 for $x0$ the procedure is as follows:

$$E_{11} * x0 * s + E_{12} * x1 * s = A_{11} * x0 + A_{12} * x1 + B_{11} * u0 + B_{12} * u1 + F_{11} * w0 + F_{12} * w1$$

Putting $x0$ alone on the left-hand side and collecting the coefficients then yields:

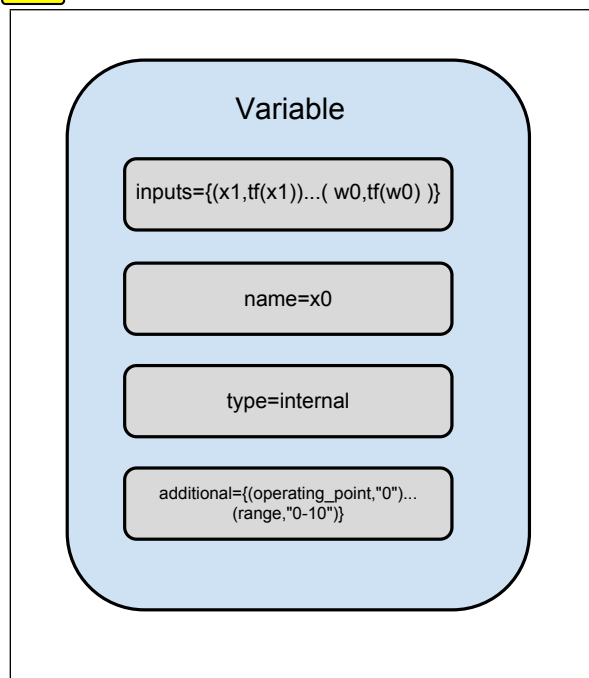
$$(E_{11} * s - A_{11}) * x_0 = (A_{12} - E_{12} * s) * x_1 + B_{11} * u_0 + B_{12} * u_1 + F_{11} * w_0 + F_{12} * w_1$$

Finally, solving for x_0 yields:

$$x_0 = \frac{(A_{12} - E_{12} * s)}{(E_{11} * s - A_{11})} * x_1 + \frac{B_{11}}{(E_{11} * s - A_{11})} * u_0 + \frac{B_{12}}{(E_{11} * s - A_{11})} * u_1 + \frac{F_{11}}{(E_{11} * s - A_{11})} * w_0 + \frac{F_{12}}{(E_{11} * s - A_{11})} * w_1$$

Naturally, only the variables with non-zero coefficients are stored as input variables.

ughly, the internal representation looks as the following:



Here **inputs** is a dictionary with variable names as keys and the corresponding transfer functions as a values. A **transfer function**, in the export tool, is using the same convention as in MatLab to represent the numerator and denominator coefficients in two separate arrays.

3.1.3 Validation and Transformations

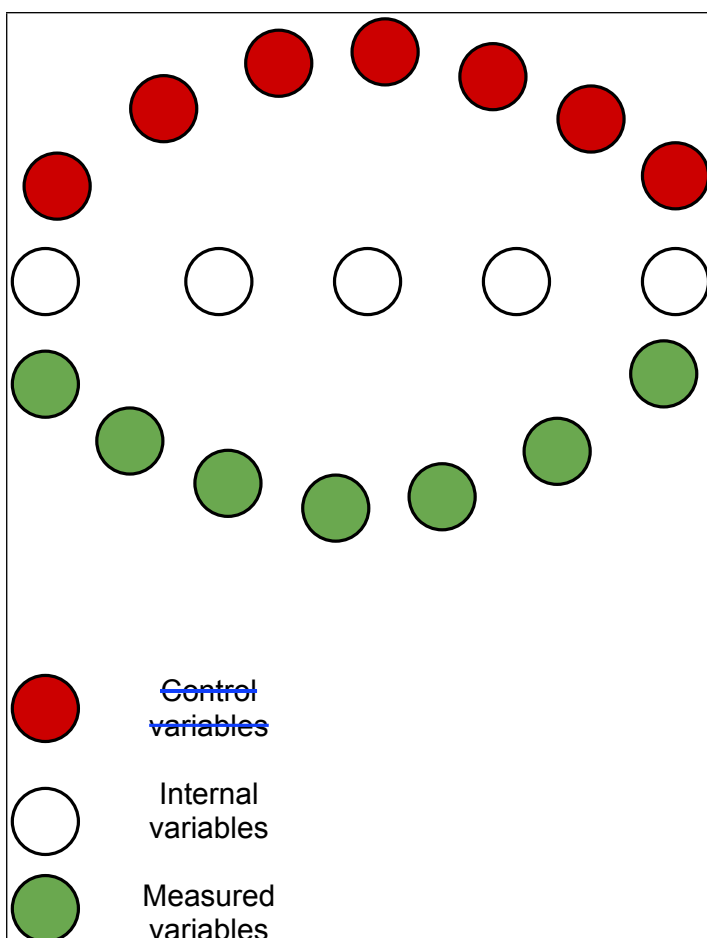
After extracting the variables, we examine the the derivatives for each of the states to make sure that they are 0, no error is produced if they are not. But a warning is emitted so that the user is aware of the fact that the model they are using probably is invalid. During this phase, additional modifications, provided by the user, should be done. Currently, the only modification supported, is the ability to indicate that some of the variables in the system should be **itted** as measured variables in the resulting ProMoVis representation. By default, all input variables are emitted as **ontrol** variables, while the others are emitted as internal variables.

3.1.4 Scenario generation

When the structure is validated and all optional transformations have been done, ~~the~~ actual generation of the ProMoVis representation is straightforward. Basically the ProMoVis representation contains two parts. First the descriptive part, a collection of all variables, with their names, graphical layout and additional mathematical properties. The second part, the processmodels, ~~are~~ describing the relations between variables in the system.

Graphical Layout

Even though Modelica supports description of position and graphical representation of models through its annotations interface, these are not necessary for a valid Modelica model. Therefore no effort has been put into trying to extract this type of meta-information. Instead, to make sure that a user of ProMoVis is able to easily access and attach controllers to the generated model the variables are emitted as depicted in the figure below.



That is, control and measured variables are emitted as the upper and lower half of a circle respectively while internal variables, that are assumed not to be of interest to the user, are collected in the center of the circle. This is achieved by a helper object, the layout emitter, that is initiated with the number of internal, measured and input variables and then calculates the step sizes in the x and y directions that will achieve the circular shape. When we then iterate over each of the variables to get their corresponding descriptive part, the layout emitter is passed as an argument, so that it can be queried for the variables graphical x and y position. This is done in a very naive way, the layout emitter has counters for each of the variable types and then, each time it is queried, it increments the current counter value together with the previously calculated step sizes to generate one x,y coordinate that is returned to the querying variable.

3.2 Interfacing with the tool

Since there are several separate phases in the tool, and each of the phases might later be extended with additional options and necessary parameters, an XML-file is used to pass information to the tool. By using XML, that is a well known, structured, format for representing data there is not only the benefit of making the inputfiles easy accessible to other developers and software, but also it eliminates the need to develop and maintain the code for parsing a tool specific format. With XML existing modules for parsing and object representation of the XML files can be used. And in the tool, an instance of the inputfile can be passed along to each of the phases. Where each phase then may just extract necessary and optional parameters needed in that particular phase.

Following is an example of the input file to the tool, containing all the necessary parameters.

```
<?xml version="1.0" ?>
<root>
  <filepath>C:\Users\Public\Documents\QuadTankPack.mo</filepath>
  <model>QuadTankPack.QuadTank</model>
  <mpattern>_pmv;_fooBar;x2</mpattern>
  <outputpath>C:\ProMoVis\output.xml</outputpath>
  <pmvoutputpath>C:\ProMoVis\QuadTank.pmv</pmvoutputpath>
</root>
```

The filepath and model nodes are the path to the original Modelica source file and the full model name, including package name respectively. The node, mpattern, contains a regular expression that the tool will try to match to the ending of the variable names. E.g. if the tool encounters the variables foo_pmv, foo_foobar, foox2 or x2 it will indicate that those variables should become measured variables in the ProMoVis representation.

3.3 DAE vs ODE

Readers familiar with Modelica and the JModelica environment might be accustomed to working with the ODE-representation of systems. The reason that the tool sticks with the DAE, even though the extraction of SFGs from an ODE is less complex, is due to the fact that when using the ODE we loose causality whenever a state depends on an algebraic variable. For a more thorough discussion of this see Appendix B.

CHAPTER 4

Result

4.1 A small example

In this chapter a small Modelica model is compiled, translated and exported to ProMoVis through the export-tool. The files used are available, together with the source code, at [GitHub\[6\]](#).

4.1.1 The input model

The model used for the example:

```
package QuadTankPack
model QuadTank
  // Process parameters
  parameter Modelica.SIunits.Area A1=4.9e-4,
                                     A2=4.9e-4,
                                     A3=4.9e-4,
                                     A4=4.9e-4;

  parameter Modelica.SIunits.Area a1(min=1e-6)=0.03e-4,
                                     a2=0.03e-4,
                                     a3=0.03e-4,
                                     a4=0.03e-4;

  parameter Modelica.SIunits.Acceleration g=9.81;

  parameter Real k1_nmp(unit="m^3/s/V") = 0.56e-6,
                 k2_nmp(unit="m^3/s/V") = 0.56e-6;
  parameter Real g1_nmp=0.30, g2_nmp=0.30;

  // Initial tank levels
  parameter Modelica.SIunits.Length x1_pmv_0 = 0.04102638;
  parameter Modelica.SIunits.Length x2_0 = 0.06607553;
  parameter Modelica.SIunits.Length x3_0 = 0.00393984;
```

```

parameter Modelica.SIunits.Length x4_foo_0 = 0.00556818;

// Tank levels
Modelica.SIunits.Length x1_pmv(start=x1_pmv_0,min=0.0001);
Modelica.SIunits.Length x2(start=x2_0,min=0.0001);
Modelica.SIunits.Length x3(start=x3_0,min=0.0001);
Modelica.SIunits.Length x4_foo(start=x4_foo_0,min=0.0001);
Real x1plusx2(start=0);
// Inputs
input Modelica.SIunits.Voltage u1;
input Modelica.SIunits.Voltage u2;

equation
  der(x1_pmv) = -a1/A1*sqrt(2*g*x1_pmv) + a3/A1*sqrt(2*g*x3)
               + g1_nmp*k1_nmp/A1*u1;
  der(x2)      = -a2/A2*sqrt(2*g*x2) + a4/A2*sqrt(2*g*x4_foo)
               + g2_nmp*k2_nmp/A2*u2;
  x1plusx2     = x2+x1_pmv;
  der(x3)      = -a3/A3*sqrt(2*g*x3)
               + (1-g2_nmp)*k2_nmp/A3*u2;
  der(x4_foo)  = -a4/A4*sqrt(2*g*x4_foo)
               + (1-g1_nmp)*k1_nmp/A4*u1;

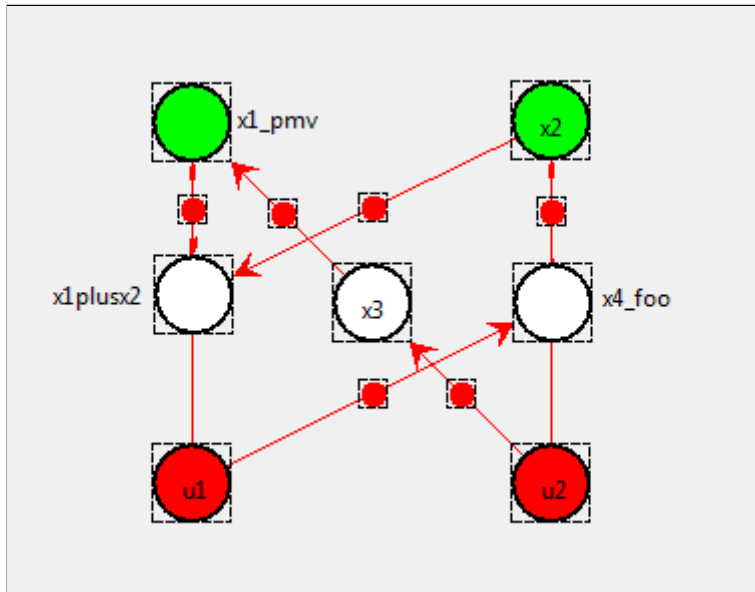
end QuadTank;
end QuadTankPack;

```

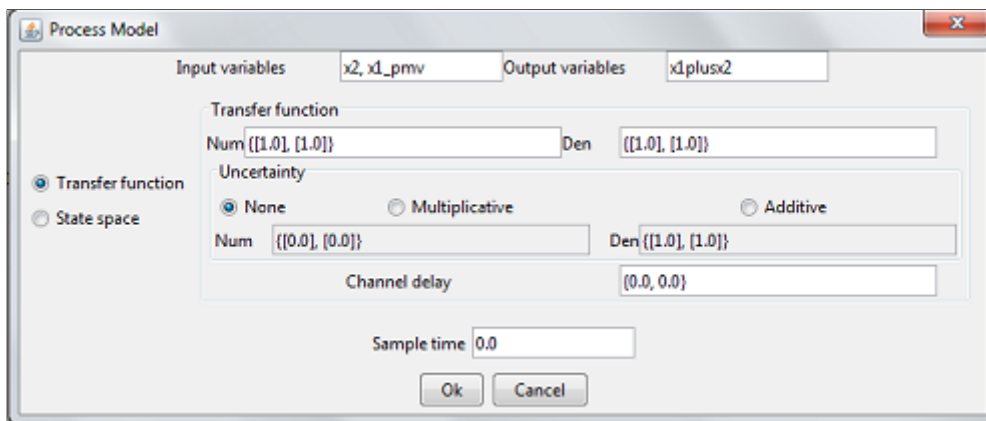
This is basically the same model as provided in the JModelica examples with some small changes. The initial values have been configured in such a way that the system is at equilibrium at time zero. As stated earlier, this is something the user has to verify beforehand with the help ~~from~~ his Modelica environment. Besides the initial values a variable, `x1plusx2`, have been added to the model. This is done entirely to introduce an algebraic variable in the system.

4.1.2 Running the export

After setup of the input file as depicted in chapter 3, indicating that variables with `"_pmv"`, `"foo_"` and `"x2"` should be set as measured variables, the `exporttool` is called. After checking that no warnings were generated, the result can be viewed in ProMoVis:

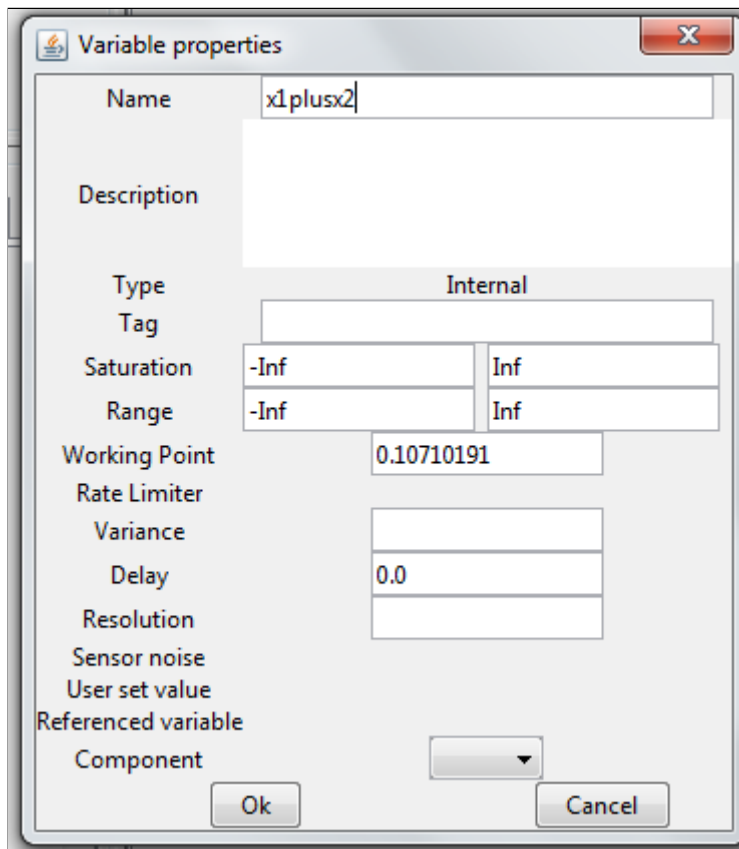


As seen, `x1_pmv` and `x2` has been set as measured variables, matching with `_pmv` and `x2` respectively, while none of the variables matched with the pattern `"foo_"`. If we start by inspecting the `processmodel` for `x1plusx2`:



Here, due to the fact that all the variables are represented as Multiple input, single output process models, it doesn't matter which of the incoming arrows to `x1plusx2` that is examined. All of the inputs, as depicted in the dialogue, are displayed. As expected, the transfer functions is unity, for both of the input variables to `x1plusx2`.

Examining the properties of the variable yields:

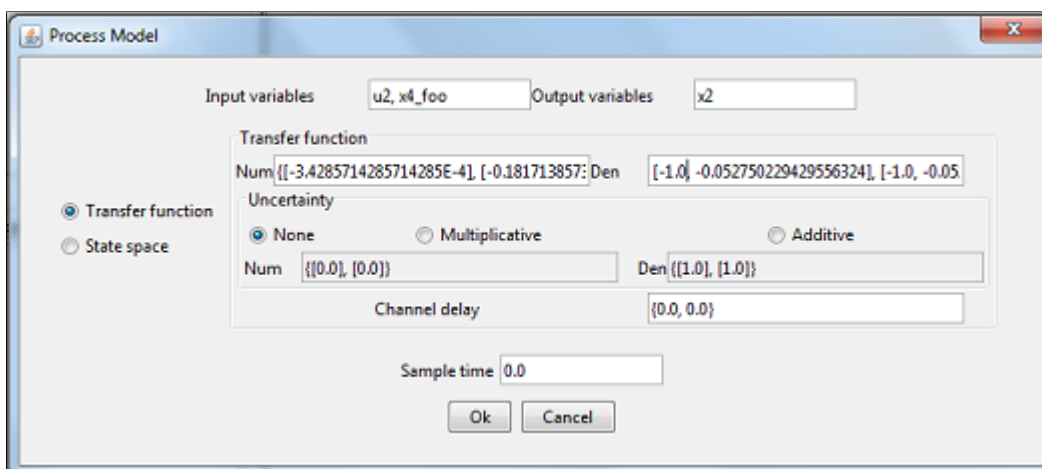


The 'Variable properties' dialog box for the variable 'x1plusx2' is shown. The 'Name' field contains 'x1plusx2'. The 'Type' is set to 'Internal'. The 'Tag' field is empty. The 'Saturation' and 'Range' fields are both set to '-Inf' and 'Inf'. The 'Working Point' is set to '0.10710191'. The 'Rate Limiter' is set to '0.0'. The 'Variance' field is empty. The 'Delay' is set to '0.0'. The 'Resolution' field is empty. The 'Sensor noise' field is empty. The 'User set value' field is empty. The 'Referenced variable' field is empty. The 'Component' field is empty. The 'Ok' and 'Cancel' buttons are at the bottom.



Since the initial values for $x1_{pmv}$ and $x2$ were set in the original model, the working(operating) point of $x1plusx2$ is the sum of the initial values.

Lets look at a variable with some more complex input relations. Choosing $x2$ it should have both $x4_foo$ and $u2$ as inputs and contain a derivative.



The 'Process Model' dialog box for the variable 'x2' is shown. The 'Input variables' field contains 'u2, x4_foo'. The 'Output variables' field contains 'x2'. The 'Transfer function' section is selected. The 'Num' field contains '[-3.4285714285714285E-4], [-0.181713857]' and the 'Den' field contains '[-1.0], [-0.052750229429556324], [-1.0], [-0.05]'. The 'Uncertainty' section is selected. The 'None' radio button is selected. The 'Num' field contains '{[0.0], [0.0]}' and the 'Den' field contains '{[1.0], [1.0]}'.

From the picture we can see that the input relations are:

$$\frac{-3.4285E-4}{-s-0.05275} * u2$$

$$\frac{-0.1817}{-s-0.05275} * x4_{foo}$$

Looking at the denominators, they are both the same, recalling the algorithm, described in Appendix A, it is realised that this will always be the case for all of the inputs to a variable, they will all share a common denominator.

CHAPTER 5

Discussion

5.1 Discussion

5.1.1 The project

Problems

A big challenge has been the lack of experience and knowledge of the mathematical tools used in modelling of physical systems. The problems have still been manageable, with the help from Wolfgang and Miguel, but i should have, in the beginning of the project, planned more time for exercising and reading in to the mathematical theory. Now, instead, i used much of the time reading in to specifications, papers and tools related to Modelica. And even though Modelica is a big part. It is, in the final implementation of the tool, a very small part and how we extract the information from the Modelica models is very dependent on the actual Modelica environment used.

Another problem, as i can see it, is that one should have tried to get in touch with someone experienced with the JModelica environment specifically. JModelica consists of many parts, written in different languages and utilizing many 3rd party tools, and sometimes it has been kind of messy to navigate through the documentation and define where the parts, relevant to this project, might reside. So a lot of time probably could have been saved if there had been a possibility to pitch some questions and design decisions against a developer or experienced user of JModelica.

These two problems together have led to many dead-ends and misunderstandings on my part, leading to a lot of additional work due to bugs and faulty assumptions about the format of the extracted data.

5.1.2 Limitations of the tool

Additional variable info

A thing that should be implemented is the ability to extract min, max, nominal and other additional parameters that a variable may have in a Modelica-model since this information would also be useful in a generated ProMoVis-representation. But right now there does not seem to be a way to extract this through JModelica. A forum post has been created, at jmodelica.org, regarding the absence of this possibility.

The graphical representation

Although the translation between Modelica and the ProMoVis-representation is now possible, the two environments [is](#) not integrated with each others in such a way that one can do modifications in Modelica, and then only the affected parts of the system is changed in ProMoVis. This is a significant drawback when it comes to the visualisation purpose of ProMoVis. A user that has imported a Modelica model into ProMoVis and done some custom layout and maybe added additional graphical representation through ProMoVis components, can't reuse this information when re-running an export, even though he might only have changed the operating points of the Modelica model.

As i can see it there exists two general approaches to this problem:

- One could add the possibility to do the export with a former ProMoVis-file as additional input. Then let the export tool could scan through the old file, look for variables with the same name and re-use the graphical information from that node. This approach still demands that the user add the graphical information needed for visualisation after the first export.
- Another approach is to add support for extracting information from graphical annotations. The graphical annotations are a standardised way, through the specification of the language [3], of creating a visual representation of a modeled system and its components.

Of [this](#) two approaches, the later should be the one to aim for since, if there is time and resources available, supporting the interpretation of graphical annotations in the export-tool could make the transition between a users Modelica environment and ProMoVis a one-click effort. Since the tool then might preserve the graphical context from the Modelica environment.

Elimination of unwanted variables

As of today, all variables present in the original Modelica model is present in the generated ProMoVis-representation. Paths in the SFG containing only internal variables

could instead be eliminated and collected into one single transfer function during the transformation phase. The challenge here is to handle loops in such a path so that the resulting transfer function is correct. The intention was to support this before the project ended, but due to lack of time this was never implemented.

**2**

Closing remarks

This project has shown that it is indeed possible to extract and translate models from Modelica into ProMoVis. For future development, in my personal opinion, the challenge does not lie in any issues related to control theory or mathematical challenges, the theory needed for this already exists and "just" needs to be implemented. Instead, the challenge lies in making it convenient for the user to use the export tool, and make it as seamless as possible to transition from the Modelica environment to the ProMoVis environment.

APPENDIX A

Extracting the SFG's in the correct order

A.1 Deciding which row to solve for which variable

When extracting the SFG's from the DAE one has to make sure that one solves each of the rows for the correct variables, making sure that an SFG can be extracted for every variable in the system. Recall the general equation for the DAE:

$$E * dx = A * x + B * u + F * w + g \quad (\text{A.1})$$

If no algebraic equations are present in the original Modelica model, the resulting DAE would have the following form:

$$E * dx = A * x + B * u + g \quad (\text{A.2})$$

The number of rows in such a system, containing no algebraic variables, would be the same as the amount of the number of declared or inferred states in the systems. The introduction of algebraic variables, that we recall are all variables that does not have a declared derivative, adds 1 row to the DAE. Since states, by definition, has a derivative declared, we can therefore look only at the E and F matrices to solve the problem of which row to solve for which variable. This since a state has to be solved for a row containing its derivative. If we declare:

$$S = [E|F] \quad (\text{A.3})$$

Here we easily realize that S will be an $n \times n$ matrix, since we by concatenating the E and F matrices, add as many columns that there is extra rows.

With the column vector L

$$L = [dx_0 \dots dx_i, w_0 \dots w_i] \quad (\text{A.4})$$

We then examine this system:

$$S * L \quad (\text{A.5})$$

By examining each of the columns in S and storing all the row numbers that contains non-zero elements together with the variable corresponding to the column a list is retrieved for each of the variables. This list then contains all the row numbers where the variable is present. The pseudocode for the process:

```
varDict.initWithKeysAndValues(L, [])
for row in S {
  for col in row{
    if(col.value !=0){
      key=L(numberOf(col));
      value=numberOf(row)
      list=varDict(key)
      list.append(value)
    }
  }
}
```

As one can see, this approach has a quadratic running time, no matter the input, compared to a pure Gaussian elimination approach, that has a cubic time complexity. The task is then to start extracting the SFG's from the original DAE with the help of this list.

If the system contains no algebraic loops, it should be possible to iterate through the dictionary and find a variable that has only one row number associated with it. By removing the found variable from the dictionary and remove the corresponding row number from the remaining variables in the dictionary. We can create a new list, solvList, in the following manner:

```
int i=0
int looped=0 // looped is used to detect algebraic loops
solvList=[]
while(varDict!=empty && looped !=2) {
  key=varDict(i).key
  list=varDict(key)
  if(lengthOf(list)==1){
    //add a tuple with rownumber and key
    rowNumber=list[0]
    solvList.append((key,rowNumber);
    //remove the key (variable), since the
    // problem is solved for this key
    varDict.remove(key)

    for otherLists in varDict{
      //we remove the rownumber
      //from the rest of the variables.
```

```
        //Since this row is already consumed
        //in the solution
        otherLists.remove(rowNumber)
    }
    looped=0;    //If we solved for a variable,
                //reset looped.
}
i=i+1;
if(i>varDict.size()){
    i=0; //restart the process from the beginning
    looped+=1; // increment looped
}
}
if(looped==2){
    putmessage( "failed due to algebraic loops")
}else{
    putmessage( "success")
}
```

At this point, we have a list of tuples in `solvList`, with each of the tuples containing a variable name and a row number, and we can begin to extract the SFG's from the DAE by solving each of the variable for the given row number.

APPENDIX B

ODE vs DAE representation of systems extracted from Modelica

Recall the general DAE representation:

$$E * dx = A * x + B * u + F * w + g \quad (\text{B.1})$$

Introducing the ODE representation that also can be extracted:

$$dx = A * x + B * u + g \quad (\text{B.2})$$

$$w = H * x + M * u + q \quad (\text{B.3})$$

Here it might be tempting to create the SFGs from the ODE instead, since one can extract the SFGs for the states and algebraic variables separately. But notice, now the states does not depend at all on the algebraic variables in w . This is still a correct system, since algebraic variables is only a combination of states, other algebraic variables and constants thus can be reduced and replaced by other states and a constant. But even though it is mathematically correct, we have lost information. The causality is broken, and a user that has declared states that depends on algebraic variables can no longer view and analyse such relations inside ProMoVis. If we consider the following, fictional, system:

$$dx_1 = x_2 + w_1 + u_2 \quad (\text{B.4})$$

$$dx_2 = x_3 + u_1 \quad (\text{B.5})$$

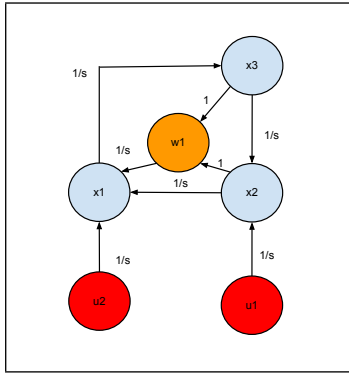
$$dx_3 = x_1 \quad (\text{B.6})$$

$$w_1 = x_3 + x_2 \quad (\text{B.7})$$

The DAE representation of this system is then:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} dx_1 \\ dx_2 \\ dx_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} w_1 \end{bmatrix}$$

And has the corresponding SFG:

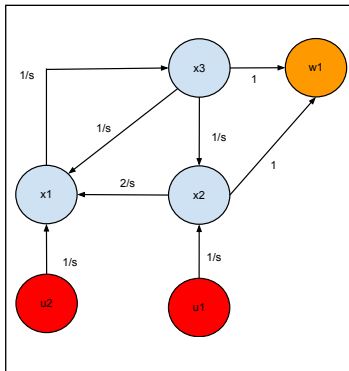


The ODE-representation of the same system would become:

$$\begin{bmatrix} dx_1 \\ dx_2 \\ dx_3 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} w_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Which then has the corresponding SFG:



As we can see, the SFG-representations of the two systems will differ, we no longer can see the relation between x_1 and w_1 . This is a problem since a user might want to

analyse the relation between the algebraic variable and other points in the system directly. FIXME SHOULD THERE BE AN EXAMPLE OF SUCH A PROBLEM LIKE A THREE WAY PIPE JOINT OR SOMTETHING

APPENDIX C

Notes from project meetings

This is the very short notes from the some of the meetings that occurred during the project. When exporting them to this report i have tried to indicate if questions and suggestions, where they occur, have been solved or not. So this is also i kind of todo list for future development of the export tool. If Anything is indicated as solved or implemented, the original report contains more information.

C.1 2012-02-27

Attending: Johan Karlsson(ProMoVis Front-end), Jesper

General discussion regarding the structure of the ProMoVis representation of the SFGs. ProMoVis does support any kind of representation of the system, one can specify the inputs to a variable as a single processmodel for every inputvariable, declare them on state space form, or you can create a MIMO-representation of the process model with several transfer functions and multiple input and output variables. The last one, the possibility to create a MIMO-representation should probably be the most convenient to use in the project, since the straight forward way to extract relations from a DAE results in a Single output multiple input model.

C.2 2012-04-02

Attending: Wolfgang, Miguel and Jesper

- Misconception about the E-matrix on my part. I cannot assume that the E-matrix will be on diagonal form. Another misunderstanding regarding what an algebraic

variable is in Modelica also implies that another assumption, that there will always be exactly as many rows as there is states is wrong. This needs to be solved.[Solved]

- Discussion around how the export tool should create the graphical layout.
 - One might emit the input- and measured variables as the edges of a circle, and then put all the internal variables, that shouldn't be of interest for the user, inside that circle.[Solved]
 - During this discussion a question came up whether or not it was possible to redo the export process, after a user might have changed the graphical layout of the system inside ProMoVis, and still keep the user defined layout. [Not solved]
 - Discussion about how the user should be able to specify which variables that should have the type "Measured" inside ProMoVis. Some suggestions were proposed but the decision was to wait for input from the discussion with the potential users.[Solved]
- lorem ipsum

C.3 2012-04-18

Attending: Jesper, Miguel, Wolfgang and Johan Karlsson

The intention was that this would be a discussion with the potential users of ProMoVis, but unfortunately they did not show up. But the following topics were covered.

- Interfacing between the export tool and ProMoVis. Here we discussed several ways to communicate between the export tool and ProMoVis, such as having a server running on localhost, running a script with arguments etc. We decided to use an XML file as input and output from the export tool. Since XML provides a well known structure and the addition of arguments and configuration options should be straight forward when/if further development demands this.
- I should implement a reference Java project that displays how one can call the export tool from Java. But should pitch some kind of interface against the user first.
- The discussion regarding the possibility for the user to modify the layout of a system and then re-import a modified Modelica model was again discussed. There is not enough time to solve this, since this is a task with many special cases that depends on the user. To accomplish this I would need time and a structured effort together with a couple of potential users.

- We continued a discussion that started over mail, regarding what should be done with the variables that is not indicated as "measured" by a user. There is the option to merge the transfer functions, but due to the possibility of loops in the paths between to "measured" variables, we initially just keep them as internal. Sticking with the graphical layout discussed in the previous meeting.
- I should make sure that there is room for interactivity between the phases. In case we would later find the need to ask the user for additional configuration options.

FIXME THERE WAS SOME INTENSE DISCUSSIONS, HAVE I MISSED NOTING NOTING SOMETHING.

C.4 2012-04-27

Attending: Jesper, Miguel and Thomas Eriksson (Optimation AB)

During this meeting we discussed the problem with Algebraic variables and how to make sure that they get valid in the generated ProMoVis representation. There was also the folloing suggestions from Thomas:

- Add the possibility to indicate that a variable in the original Modelica model should be of the type "Measured" in the generated ProMoVis scenario. Prefferably through the matching of the last part of the variable names.[Solved]
- Try to extract the max and min values, if specified in the Modelica model and use these as values as "Saturation" in the ProMoVis representation. [Not Solved]
- If nominal values is specified, use 0-nominal value as "Range" in the ProMoVis representation.[Not Solved]
- See if the scaling factors can be extracted and try to find a way to automatically identify the tolerances on the derivatives with the help from this. E.g. are they "close enough" to zero in the linearised model.[Not Solved]
- Thomas is going to send som "real" models that we can test in the export tool.

REFERENCES

- [1] W. Birk, M. Castaño, A. Johansson, and S. Rönnbäck, *INTERACTIVE MODELING AND VISUALIZATION OF COMPLEX PROCESSES IN PULP AND PAPER MAKING*. 2010.
- [2] J. Åkesson, K.-E. Årzen, M. Gäfvert, T. Bergdahl, and H. Tummescheit, *Modeling and Optimization with Optimica and JModelica.org-Languages and Tools for Solving Large-Scale Dynamic Optimization Problems*. Elsevier, 2010.
- [3] M. Association, *Modelica language specification 3.2*. 2010.
- [4] “Jmodelica.org,” June 2012.
- [5] “scipy.org,” June 2012.
- [6] “Github.com/jwmborg/modelica-to-promovis-export/,” June 2012.