Modelica Simulator Compatibility - Today and in Future

Jörg Frochte
Hochschule Bochum
Department of Electrical Engineering and Computer Science
Höseler Platz 2, 42579 Heiligenhaus, Germany
email: joerg.frochte@hs-bochum.de

Abstract

In this paper we would like to give a small snapshot in time on Modelica tool compatibility today, and discuss strategies for its improvement in order to keep it on a high level. Especially we would like to consider approaches for semi-automatic test and verification frameworks as well as to develop different levels and definitions on Modelica tool compatibility.
Keywords: Modelica language; simulation and design tools; quality and compatibility management

1 Introduction

The Modelica language grows as well in complexity as in scope and becomes a mighty tool to describe models from very different domains. Beyond this the number of tools using the Modelica language has increased to the benefit of the users. A various set of compatible simulators decreases the dependency on a single tool provider, allows exchange between different modelers using different tools, and – because every development of a model and its maintenance is not for free – it offers a high degree of investment security.

But there are dangers to be avoided as well. A diverging interpretation of a standard and a heterogeneous set of vendors may lead to unpleasant scenarios for users and library providers.

Compatibility is not an easy task for a complex declarative language which is the base of model-based software generation. Beyond the language and its interpretation the results given by the generated code may differ based on various parameters like the chosen integrator and its boundaries for relative and absolute errors. So some variations in the results are allowed and expected, some are not.

Drawing a comparison with C++-Compilers it is expected that the quality of the generated executable differs, e.g. concerning performance. It might be accepted that not every compiler can handle any part of the new C++ standard, but it falls beyond the pale, if source code is successfully compiled and executed, and finally provides totally different results based on the used compiler.

In order to receive an impression how compatible Modelica tools are among themselves and as well among the standard we took a snapshot of four tools, mostly demo and testing versions, and gave them quite tiny models that can be translated with limited versions.

For us the fact which tool can handle what kinds of models is of minor interest, because this will properly change with every new version. More important for us is the answer to the question if there is already a diverging interpretation of the standard or if this issue has been of no interest so far. In this case we could assume that there will probably be a growing need for measures guaranteeing quality as well as compatibility, which have to be presented later on.

2 Snapshot on Modelica tool compatibility

Let us start with a scenario in which two developers are using different tools to model and simulate. Now developer A and B exchange models, given in Modelica source code, and naturally they both expect no problems, because they meet the Modelica standard. Because of the grown complexity and increasing number of interpretations of the standard this may run ill and so it may turn out that an easy exchange is not possible. We distinguish between three different kinds of causes presented in the following sections.

2.1 Lacking support of language elements

The lacking support of language elements is easiest to handle for the users. In this case the translation of the Modelica model will fail and provide a hint which language element is not supported. In a

lot of Modelica tools a common non-provided language element is the *else-when*-element in the equation section of a model. This type of compatibility problem is annoying but it causes no danger.

A good example is the following model:

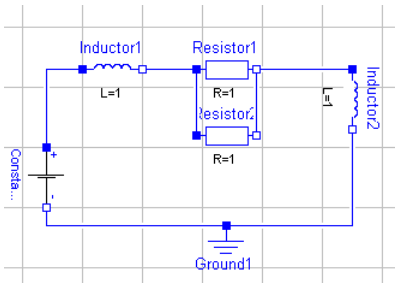
```
model A
  Real x;
  Real a(start=2);
equation
  when {time > 0.5,x > 0.7} then
    if time > 0.55 then
      a = 3;
    else
      a = 4;
    end if;
  elsewhen time > 0.6 then
    a = 5;
  end when;
  der(x) = 1;
end A;
```

Model 1: else-when in equation section

It turned out that in our snapshot of four Modelica tools only one could translate and simulate the model above. Two of them were not able to translate it and one generated code but directly aborted using the compiled program.

2.2 The generated code

The next cause has its roots in the different skills of the tools to optimize code and handle tricky situations concerning e.g. index reduction. Most tools use a more or less straightforward implementation of the Pantelides Algorithm [4], extended with dummy derivatives (s. e.g. [5] or [1], chapter 7). In some situations this is not always enough and may lead to non-executable results. For example, this combination of inductors and resistors cannot be simulated in every Modelica tool:



Model 2: advanced index reduction

These kinds of compatibility problems are in a way in between, if they are real compatibility issues at all. The quality of code generation and GUI is the reason for a user choosing one tool or another. So on one side diverging results in quality should be expected and respected, but in any case it would be interesting to measure it. On the one hand the tool provider is presumably keen on increasing the quality of his product. And on the other hand, if the results were published, it may even help the customer to choose a Modelica tool. So we think these are the kinds of variations in the results that are allowed, expected, and do not touch the goal of standardized and compatible language.

However, if the code is executable and runs without a warning, which was not the case for any tested tool, this scenario comprises some risks. So like it seems to be now, a termination of the simulation as soon as possible should always be the default handling of such cases. Thus we assume that in most cases the simulation will be directly aborted if such a problem arises.

2.3 Different interpretation of language element

In contradistinction to lacking support of language elements or variations in the code quality a different interpretation of language elements of the various Modelica tools may lead to more serious problems. Let us have a look at the following example:

```
class A
  class C
    Real t(start=-2);
    Real x;
  equation
    der(x) = t;
  algorithm
    when time > -0.1 then
      t := time + 0.1;
    end when;
  end C;
 C c;
end A;
```

Model 3: when and HDAE initialization (I)

All of the provided examples are very small and can be evaluated in most test or demo versions of Modelica tools. From a theoretical point of view and our interpretation of the Modelica standard [5] the initial value of *t* should be *-2*. In our test it turned out that just one of the tested Modelica tools computed the result *-2* and most of them started with *t = 0.1*. Independent of the correct value, the interesting effect for us is that the whole simulation may run differently now. In the next model, the problems might

have their cause in interpretation or implementation of initial conditions and equation reduction.

```
model A
 Real x(start=7);
 Real y,z;
 flow Real a[2,3];
equation
 z = -a[2,3];
 z=x;
 y=z;
 a[1,1] = 0;
 a[1,2] = 0;
 a[1,3] = 0;
 a[2,1] = 0;
 a[2,2] = 0;
 der(a[2,3]) = 1;
end A;
```

Model 4: Transfer of start values

In our interpretation the resulting equation *x=y* should propagate the initial value, so that the simulation starts with x=y=z=7 and a[2,3]=-7.

Our point of view is that a consistent initial value should always be tried to be propagated directly, independent of an additional attribute like *fixed=true*.

But in some tools start values are not propagated and so the simulation starts with different initial values. The effect is of course a simulation with totally different results. A possible explanation might be that some tools call the initial function with x=y=z=7 and a[2,3]=-7 and try to find a consistent set of variables based on these initial values. Whereas the other tools start with this procedure with x=7 and assume that this information will be propagated during the DAE initial problem.

Finally let us look at the following very small model 5:

```
model A
  Real x(start=2);
  Real y(start=3);
equation
  der(x) = 1;
  when x > 1 then
    y = pre(y) + 1;
  end when;
end A;
```

Model 5: when and HDAE initialization (II)

In the Modelica Standard 3.2 ([5]), section 8.3.5 the language element is defined as follows: "The statements within a *when*-equation are activated when the scalar expression or any of the elements of the vector expression becomes true." So it could be translated in an *if*-condition like this:

if (boolean) and not pre(boolean)

In combination with the techniques for an initialization after an event indicated on page 226 of [5] - the init situation is slightly the same - we come to the conclusion that the condition should only be activated, if *x>1* has been logical false before and is now logical true. This situation is quite tricky for a lot of Modelica tools. We think that *x* is equal to *2* immediately, and the state x has a positive derivative during the whole time. Therefore, the *when*-cause should never be activated and so *y* is equal to *3* all the time. Just one Modelica simulator handled the situation like this, all the others computed *y=4*.

So we have got two situations: model 3, in which the time value has to be interpreted, and model 5, which includes an initial value for x. In both situations the when-language element is misinterpreted during the initial phase. In case of model 5 one might argue about the start value of x, but obviously the time value as in model 3 is never negative.

We tested some more models, and as well as in the presented ones we found a few similar results. So we can conclude that there are diverging interpretations of the standard and that it makes sense to think about strategies in order to avoid this.

3 Suggestion of semi-automatic test and verification frameworks

The first step in our strategy is briefly presented in the following figure:

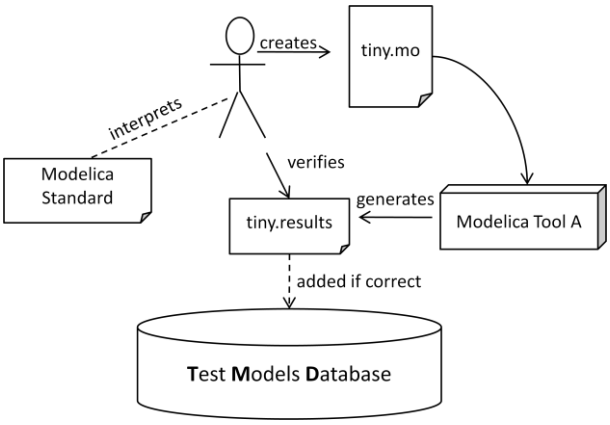


Figure 1: Test Database and models with proved results

The most important aspect could be to create a database with tiny Modelica models which contain a single element of the Modelica standard one would like to have tested. The models themselves should be tricky for the translation process, but so small that the results can be exactly verified by a human based on the common interpretation of the Modelica standard. This is very important because certainly it is not proved that Modelica tool A in the figure is free from errors.

These tiny models are in a way academic and their benefit is that the correctness of the results can be proved. When a correct pair of model and result has been created, both are added to the database.

This is the most important aspect for a check for compatibility. But it is hard to collect a lot of these models and it might not be possible to get all complex side effects. So beyond this, one should add "applied models" to the database:

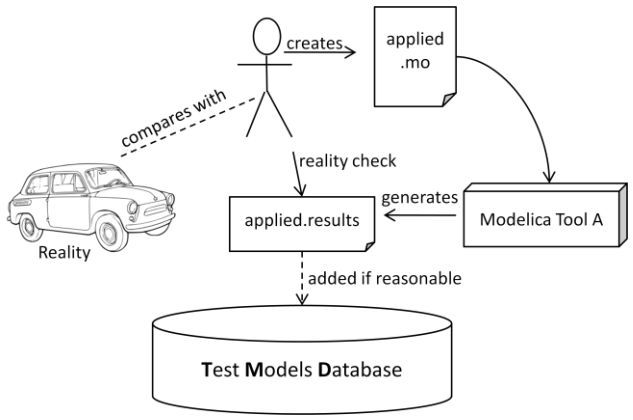


Figure 2: Test Database and applied models with reasonable results

In general, no human can prove the correctness of a complex model like this. So in this case it is just a check if the results are reasonable or not. If they are reasonable the reference results as well as the model are added to the database. In the next step it makes sense to distinguish between these different kinds of models:

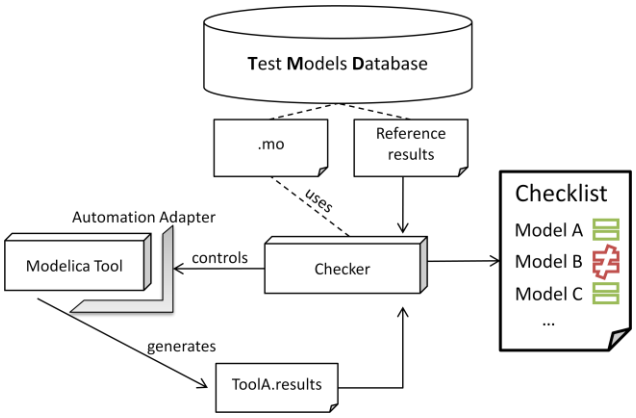


Figure 3: Semi-automatic test scenario

With this database it would now be possible to test the Modelica tools on Standard compatibility at least semi-automatically. A checker program can pick up the Modelica model with its associated reference results, simulate the model with the Modelica tool under test and compare the results. The comparison of the results needs a little bit a fuzzy approach, because it is unlikely that the generated code will produce totally the same results as in the reference solution. There are a lot of interesting approaches, see e.g. [8], to compare models and their results in a (semi)-automatic way. For simplicity let us assume that all variables of the model are stores in a vector $\boldsymbol{x}$ and the corresponding reference solutions in a vector $\boldsymbol{r}$, then one might check for $|\boldsymbol{x}_i(t) - \boldsymbol{r}_i(t)| < tol$. Another important benchmark value should be the time of the event as well as the initial values after the event.

If every compiled Modelica code uses a different integrator for the HDAE, this might lead to serious problems for the compression, so just for this benchmark purpose one should fix a freely available Open Source integrator like IDA (s. e.g. [3]). With a fixed integrator such a test should be possible.

But why do we call this a semi-automatic test for it looks like a full-automatic test until now? Responsible for this are the failures which may occur when simulating or translating the "applied models". In this case it is not possible to exclude the possibility that the reference results are wrong. So in some cases it might be necessary to reevaluate the reference results for such a model.

4 Levels of compatibility and quality control

With such an infrastructure a validation service could be set up. Nevertheless, this would just produce a list of successfully and not successfully handled models, but no solution to the fundamental problem. So at that point one would be able to measure the language respectably the tool compatibility and achieve information about what to improve. But this is just one aspect of two. The two challenges for the compatibility issue on the long run are firstly the growing complexity and secondly measurement and control. But up to now we majorly dealt with the measurement aspect. How could growth of complexity be reduced without acting as a brake upon new innovations in the Modelica language?

A possible approach might be introducing different levels of complexity in the Modelica language. The highest level could be today's Modelica language with all its language elements as well as upcoming innovative and progressive features. The lower levels should be becoming more and more conservative concerning changes and less complex:

1. (Full) Modelica
2. Simple Modelica
3. Flatmodelica

One might think that Flatmodelica already exists for very often the term "flat model" is mentioned, and therefore one has an association, because e.g. some tools allow exporting "Flatmodelica" or a "flat model". But this is not true, for there is no official standard defining Flatmodelica or at least a "flat model". What comes close to a definition is written in [2], chapter 18, but anyway it is not a formal language description. The name suggests that there are no more dependencies concerning libraries. Beyond this, it implies that no more inheritance has to be carried out, but this is hard to be done, because nearly everything in Modelica is a class. So e.g. the question occurs whether records are part of Flatmodelica or not.

Maybe we should motivate why it could make sense to introduce different levels of complexity and development speed in the Modelica standard.

The major reason could be that no tool has been able to implement the full Modelica language up to now. If the language keeps on growing, as during the course of recent years, it is hardly probable that this status will change.

To illustrate this, let us have a look at the following model:

```
class A
  Real x[2, 3];
  Integer i=7;
  Integer j=8;
  Real y;
equation
  for i in {1,2}, j loop
    x[i, j] = i*A.j;
  end for;
  for i in {3,4}, j loop
    x[i, j] = A.i*j;
  end for;
  y=time*x[2,2];
end A;
```

Model 6: Automatic detection of array bounds

This model exclusively uses valid Modelica language, but in our tests none of the tools has been able to generate code and simulate it successfully. The used language elements are not new in Modelica 3.2, so we can exclude the effect that the tools were unable to implement them in time. An explanation might be the high demands of such dynamic language elements for the data structures of a Modelica tool. This model is not an isolated incident; let us for example regard this model:

```
model A
  record R
    Real x[1,1];
    Boolean b;
  end R;
  Real w;
  R r1(x={{1}}, b=false);
  R r2(x={{2}}, b=true);
  R r3[2];
equation
  r3[1] = r1;
  r3[2] = if time > 0.5 then
            (if time > 0.6 then r1 else r2)
           else r2;
  der(w) = r3[2].x[1,1];
end A;
```

Model 7: Usage of records

Records as well as the automatic detection of array bounds and especially their dynamical handling cause just one problem in many tools. The effects probably differ because of the different data structures used to translate the Modelica models.

So we can conclude that full Modelica, as fast developing language standard, is not predestinated as cross-tool exchange language. Beyond these features and aspects, there are a lot of chapters in the Modelica standard that could likely be excluded for a Simple Modelica approach. Examples might be [5], section 10.5.1 “Indexing with Boolean or Enumeration Values”, chapter 14 “Overloaded Operators”, some of the redeclaration features described in chapter 7.3 or the expandable connectors from chapter 9.3.1.

Such features proposed to be excluded from Simple Modelica in comparison to full Modelica are mostly the very dynamic ones and therefore hard to be validated using the proposed semi-automatic test and verification framework, especially but not only concerning more complex models. In fact the suggested test and validation infrastructure will be more efficient on the lower levels and always less efficient on the higher ones. One reason would be the conservative progressing approach and another one limited amount of language elements and features.

Because the mentioned features are apparently difficult to be implemented for Modelica tool vendors, it would probably be possible to achieve a better tool compatibility on the lower levels compared to the higher ones. Beyond this, the introduction of less featured levels of Modelica might even lead to a provision of a base for the exchange of Simple or Flatmodelica models to and from the proprietary Simscape language, s. e.g. [8], invented by TheMathWorks. To support such a scenario for Simple Modelica the redeclaration techniques described in [5], section 7.3, might need further restrictions or simplifications. Anyway it is of course unlikely that this will work without a conversion procedure, but a conversion from Simple Modelica to Simscape might be possible, while the more complex and mighty Full Modelica is a formidable challenge for an automatic conversion from and majorly to Simscape.

So Simple Modelica and Flatmodelica as subsets of full Modelica could provide grand strides concerning cross-tool exchange, tool compatibility and finally make formal tests discussed in section 3 much more efficient.

If we had these two subsets of Modelica we could judge tools by their capacity to import, export and translate models on three different levels. To achieve a simple measurement for users one may introduce a bronze, silver, gold and platinum tag on the different levels. The platinum tag will just be given, if a tool can handle the full test without failures, the gold tag with a given percentage and so on. The suggested infrastructure could be set up by a central organization like the Modelica Association.

As discussed above it is very unlikely that a tool would reach the platinum level for the latest few full Modelica language versions. Most tools could achieve gold and silver, but obviously this would not be the perfect exchange level because every tool might miss different language aspects. On the more conservative lower language levels like Simple Modelica or Flatmodelica a lot of tools could reach platinum level and therefore provide a good base for a cross-tool exchange with a consistent language interpretation.

For a kind of “flat model”, like e.g. in Dymola, can be generated from full Modelica without any loss of functionality it should be possible to do the same with a formal defined Flatmodelica and a Simple Modelica. So there won’t be any loss of functionality, just a loss of structure and convenience. This is the reason why Simple Modelica as intermediate stage between full Modelica and Flatmodelica makes sense. Flatmodelica as kind of textual description of an HDAE is always possible, but it is hard to maintain a model described in Flatmodelica while it is hard to achieve a high compatibility level for full Modelica. So Simple Modelica together with a semi-automatic test and verification framework could lead to a high degree of investment security and independence for users and library providers.

5 Conclusions

So finally we conclude that the desirable growth of Modelica tool vendors and language capacity leads to a lot of benefits but also to the issue of compatibility which today and in future will become more and more important.

We have shown that recently there is a need to introduce quality control mechanisms. Beyond this, we tried to give brief suggestions how it might be possible to deal with the task of compatibility of Modelica tools among themselves and as well among the Modelica standard.

References

[1] F. E.Cellier und E. Kofman. **Continuous System Simulation**, Springer (2006)

[2] P. Fritzon. **Principles of Object-Oriented Modeling and Simulation with Modelica 2.1**, John Wiley & Sons (2004)

[3] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker and C. S. Woodward: **SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers**. In ACM Transactions on Mathematical Software, 31(3), pp. 363-396, 2005.

[4] Mattsson and Söderlind. **Index Reduction in Differential-Algebraic Equations using Dummy Derivatives**, SIAM J. SCI. COMPUT. Vol.14. No.3, pp. 677-692 (1993)

[5] *Modelica Association*: **Modelica Language Specification Version 3.2**; (March 2010)

[6] C. C. Pantelides. **The Consistent Initialization of Differential-Algebraic Systems**. In SIAM J.Sci.Stat.Comput. Vol.9, No. 2, (1988)

[7] B. Stein. **Model Compilation and Diagnosability of Technical Systems.** In 3rd Int. Conference on Artificial Intelligence and Applications, pp. 191-197 (2003)

[8] *TheMathWorks*: **Simscape 3 Language Guide** (March 2010)