An External Model Interface for Modelica

Torsten Blochwitz Gerd Kurzbach Thomas Neidhold
ITI GmbH, Webergasse 1, 01067 Dresden, Germany

Abstract

The paper describes the integration of non-Modelica submodels to a complete Modelica model. We show, that the Modelica standard interfaces to external code (external function and external object) are not suited to integrate the behavior of non-trivial models. The necessary enhancements of the external object interface are worked out and the usage is demonstrated.

Keywords: External Function, External Object, C-Interface

1 Introduction

With ITI-SIM and SimulationX [1] the company ITI develops and distributes software for system simulation since 1991. SimulationX provides full support for Modelica since release 3.0. The steadily growing acceptance of these programs is based on a modern user interface, which enables engineers an easy access to modeling, simulation and optimization techniques by using efficient calculation methods associated with a wide range of libraries and tools. A large contribution to this success is the availability of interfaces to other CAE tools like MATLAB/Simulink, MSC.ADAMS or SIMPACK. In addition to various forms of co-simulation the C code based exchange of models between different tools is also supported. This enables the user to cooperate across team boundaries independent of the finally used simulation tools. The encapsulation of the model functionality, which will be achieved by the compilation of the code generated from the original model, also allows an effective protection against unwanted insight into the parameters and behavior. With the description of this interface, as well as our proposal for its integration into the Modelica language we want to make available the described advantages to the whole Modelica community.

2 Motivation

There are different motivations to integrate non-Modelica submodels into Modelica models:

1. Sometimes a component is modeled using a specialized simulator for a specific physical domain (e.g., SIMPACK for complex multi body systems or GT-POWER [2] for combustion engines). For system simulation within a Modelica simulator the component should be integrated into a Modelica model. Often the model functionality of the special simulator can be exported as C-code.
2. A supplier has developed a model of a component in Modelica. He wants to supply this model to the OEM but wants to protect his know how, contained in the physical model. The safest way to do that is to provide the model in binary form as a compiled library with a well defined interface.

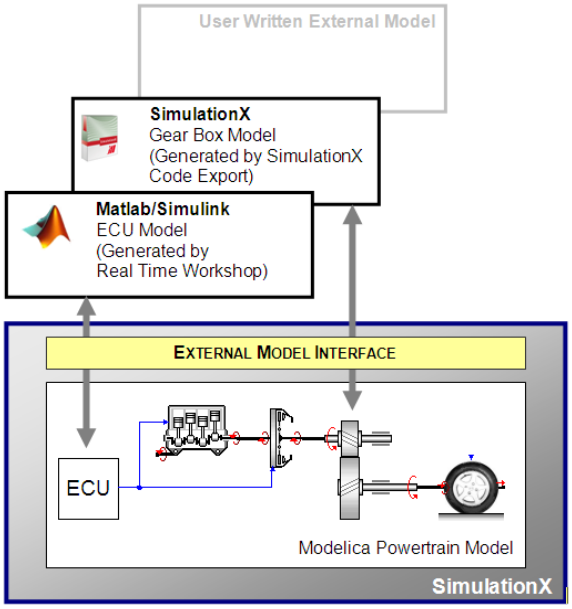


Figure 1: Modelica model with embedded external components

Modelica currently supports the following interfaces to external functionality [3]:

- external function interface
- external object interface

According to the Modelica Language Specification results of external functions may only depend from their arguments, i.e., the functions have no internal memory. Complex models do have a memory.

External objects as an improvement of external functions provide a memory context which is reported between the function calls.

Beside the more or less complex function for the right hand side of an ODE or DAE, external models may contain discrete states, state- or time-dependent events, or delay buffers. To integrate those into the simulation, information about the objects have to be exchanged between the external model and the simulation environment. The Modelica external object interface does not provide the functionality to exchange this information. It must be extended to an "External Model Interface." The following chapter describes the requirements to the external model interface resulting from the features of complex models.

The inclusion of controller code (e.g. ECU code generated by the Real Time Workshop from The MathWorks) is not subject of this article. Such components must be called with a constant sample rate during the simulation. This can be done utilizing the existing Modelica interfaces (external function or external object). No extensions are necessary.

3 Requirements for External Model Interface

3.1 Requirements Resulting from Model Features

At first we consider external models, which are represented by ordinary differential equations (ODE). The equations may contain discontinuities.

Such systems are represented by following equations:

$$\dot{x} = f(x,u,p,z,r,t) \quad (1)$$

$$y = g(x,u,p,z,r,t) \quad (2)$$

$$z = h_1(x,u,p,r,s,t) \quad (3)$$

$$r = h_2(x,u,p,z,t) \quad (4)$$

with:

x ..... Continuous states
u ..... Inputs
y ..... Outputs
p ..... Parameters
z ..... Discrete states
r ..... Root functions
s ..... Sample variables
t ..... Time.

Equation (1) represents the right hand side (RHS) of the ODE. Equation (2) represents the calculation of the outputs. Both calculations should be separated in different functions to enable an optimum arrangement of the external model in the calculation sequence of the enclosing model.

The other equations deal with event handling and discontinuities.

Events:

Two kinds of events must be handled: time events and state-dependent events. Time events are produced by timers or the Modelica `sample` keyword. They are signaled from the solver to the model by setting corresponding sample variable s.

State events are signaled from the model to the solver by zero crossings of the root functions r. Discrete variables z can change its values only at events.

According to our experience a reliable event handling is crucial for a robust and fast calculation.

Reinitialization of States:

At event instants state values may be reinitialized by the external model. The solver should be informed about such an operation.

Additional Model Information:

External models of specific domains may provide further information which eases a robust and fast solution. Examples are minimum and maximum permissible values for states (e.g. absolute temperatures and pressures have to be positive).

Other models could provide the Jacobian matrix directly.

It depends on the simulator, if this data is used.

Special Features:

Some special features demand actions on valid model data. For example, the buffers of delay blocks must be updated with valid data once after a successful time step. For that reason, the external model must be called once after successful steps with valid data and must be informed about that.

Other model features may require the allocation and freeing of memory or data is to be read from files once. For that reason special functions must be called once at the beginning and the end of the simulation run.

3.2 Requirements Resulting from the Integration into the Enclosing Model

For integration of the external model into the enclosing Modelica model the external model calls must be correctly positioned in the calculation sequence.

If the outputs of the external model depend only from states, the arrangement is simple: the external model must be called before one of the outputs is needed.

If the external model has direct feed through (outputs depend directly from inputs) the situation is more complex. The external model must be called before the outputs are needed and after the inputs are calculated. If the enclosing model defines dependencies of the inputs from the outputs of the external model, we have algebraic loops. The simulator must treat them in an appropriate manner.

For this reason it is essential for the external model to provide the information, which output depends on which input(s). If the model creator is not able to offer this structural information, the worst case (each output depends from each input) has to be assumed.

3.3 Technical Requirements

The external model interface for Modelica should be similar to the Simulink S-function interface from The MathWorks [4]. This interface is quite well adopted and widely used.

The realization of the data transfer should be simulator-specific. The external model accesses the data via functions or macros. These functions or macros are provided by the target simulator.

The external model interface should be usable by non-Modelica simulators too. These simulators should be able to use and/or to create models using the interface.

We will assume that at least the interface part of external models is written in C. How the external model is linked to the simulator is tool specific and depends on the capabilities of the operating system.

4 The External Model Interface

The external model interface can be seen from the following three perspectives:

- Specification of the functions and data provided by the external model.
- Specification of the calling sequence by the solver.
- Specification of the interface to Modelica.

These three views to the external model interface are shown in Figure 2.

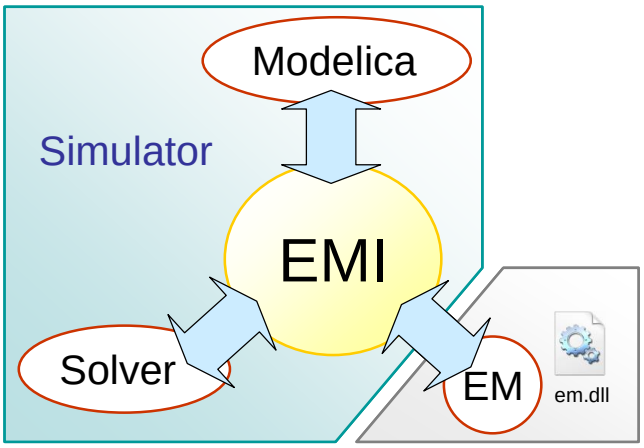


Figure 2: Three views to the external model interface

On the other hand the interface provides a set of utility functions which can be called from external model code.

According to the requirements we get the following data flow between the components (Figure 3).

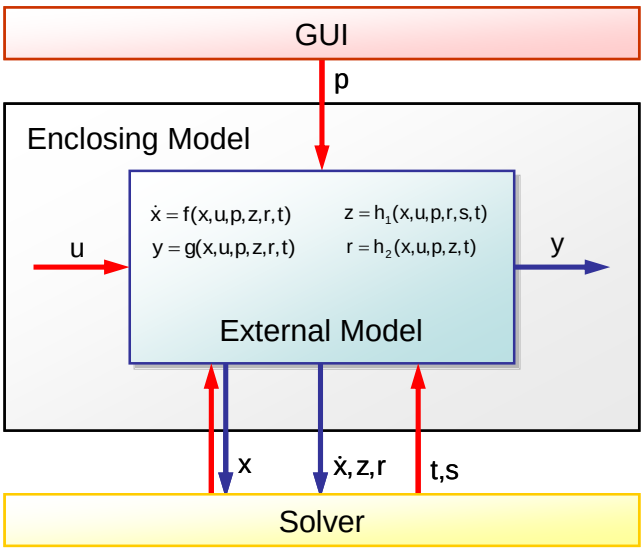


Figure 3: Data flow

The details, i.e. which data is to be provided by which function, are part of the complete specification, which will be published by the authors.

4.1 External Model View

The data transfer is realized via the external model context, the structure `emc`. The external model must implement the following functions:

`void emiInitializeSizes(emc *C)`

- Defines the dimensions of the model.
- Transfers additional information (input – output dependencies)
- Is called multiple times before the calculation.

`void emiStart(emc *C)`

- Is called once at the beginning of the simulation run.
- Can be used, e.g., to allocate memory.

`void emiInitializeSampleTimes(emc *C)`

- Transfers constant sample times.
- Is called once at the beginning of the simulation run.

`void emiInitializeConditions(emc *C)`

- Sets the initial conditions for continuous and discrete states.
- Is called once at the beginning of the simulation run.

`void emiTerminate(emc *C)`

- Is called once after the simulation run.
- Allocated memory can be freed here.

The following functions are called *multiple* times during one calculation step:

`void emiDerivatives(emc *C)`

- Computes the RHS of the ODE **(1),** and **(3)** during event iteration.

`void emiOutputs(emc *C)`

- Computes the outputs **(2)**.

`void emiZeroCrossings(emc *C)`

- Computes the root functions **(4)**.

The next function is called *once* after a successful calculation step:

`void emiUpdate(emc *C);`

- Called after a successful calculation step with valid data.

It is not allowed to access the data in the external model context `emc` directly. Instead, a set of function or macros is to be used, e.g.:

`emcSetNumContStates(emc *C, int_T n)`

- Sets the number of continous states.

`emcGetContStates(emc *C)`

- Returns a pointer to the state array.

`emcSetSolverNeedsReset(emc *C)`

- Informs the solver about a reinitialization of states.

The implementation of the `emc` and the access functions are target tool specific and must be provided by the simulator manufacturer.

4.2 Solver View

Figure 4 shows a simplified flow chart of the solution process for a Modelica model. It demonstrates which functions are called at each stage.

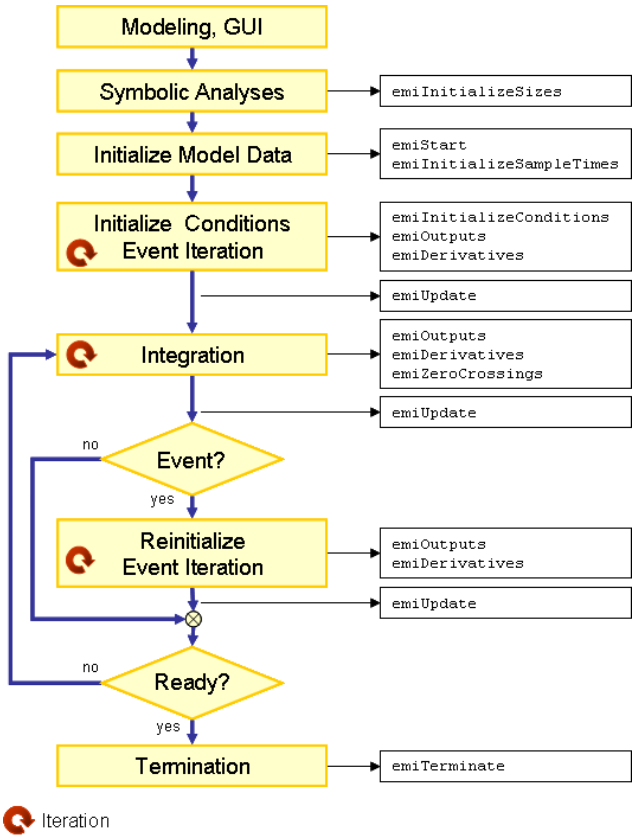


Figure 4: Solution process flow chart

If the integrator works iteratively, the functions `emiOutputs` and `emiDerivatives` may b called several times at the same time be instant with temporary data. These functions are to be implemented as reentrant and must not store any data.

For these purposes `emiUpdate` is called with valid data once after a successful time step.

The method for robust handling of discrete variables during event iteration is an open issue at the moment. There are several possibilities, which should be discussed with other simulator vendors.

4.3 Modelica View

This section describes the enhancements of the external object call interface to the external model interface. The information to be exchanged between the external model and the Modelica simulator are of two types:

- Data for the model (parameters, inputs, outputs). These are exchanged via usual function arguments and appear inside the Modelica model.
- Data for the solver (states, derivatives, residuals, discrete states, root functions…). These are handled implicitly by the simulator using the external model context.

We suggest the new Modelica built in type "external model" as an extension of the external object interface. The implicit declaration of the type could be:

```
class ExternalModelInterface
  extends ExternalObject;
    function constructor
      input String emName;
      output ExternalModelInterface emi;
      external "C" emi=initEM(emName);
    end constructor;
    function destructor
      input ExternalModelInterface emi;
    external "C" terminateEM(emi);
  end destructor;
end ExternalModelInterface;
```

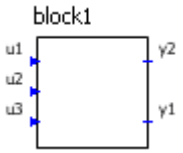
The calculation function is declared implicitly as follows:

```
function calcEM
  input ExternalModelInterface emi;
  input Real u[nu];           //inputs
  input Parameter Real p[np]; //parameters
  output Real y[ny];          //outputs
  external "C" y=calcEM(emi, u, p);
end calcEM;
```

Differing from the external object interface, the functions `initEM`, `terminateEM` and the calculation function `calcEM` do not correspond one to one to the functions of the external model. During the symbolic analyses of the model these functions have to be mapped to the appropriate function calls of the external model.

The dimensions (`nu`, `np`, `ny`) and the dependencies of the outputs from the inputs must be known during the symbolic analyses. This information should be provided by the external model. To get this information, the external model must be called already during the analyses. This is another difference to the external object interface.

The usage of the external model interface in a Modelica model is:




```
model Block "Block with External Model"
  input SignalBlocks.InputPin u1;
  input SignalBlocks.InputPin u2;
  input SignalBlocks.InputPin u3;
  output SignalBlocks.OutputPin y1;
  output SignalBlocks.OutputPin y2;
  ExternalModelInterface emi=
    ExternalModelInterface("c:\test.dll");
  equation
  {y1,y2}=calcEMI(emi,{u1,u2,u3},{1,2,3});
end Block;
```

As denoted before, the Modelica model handles only the inputs, outputs, and parameters of the external model. The other information is exchanged implicitly between the solver and the external model. If the user wants to access such internal data for debugging purposes, special functions could be provided. Access to the states could be given by:

```
function getEMStates
  input ExternalModelInterface emi;
  output Real x[nx];      //states
  external "C" x=getEMStates(emi);
end getEMStates;
```

5 Application Scenarios

5.1 Hand-Written External Models

External models can be developed by any programmer. The complete API with all necessary data structures and functions is described in a programmer's manual. Normally it should be the exception to implement an external model completely by hand. Instead, the adaption and integration of existing source

code according to the external model interface requirements will be the typical task. This way is practicable for single solutions and non-commercial applications. The necessary work can be simplified by using precast templates.

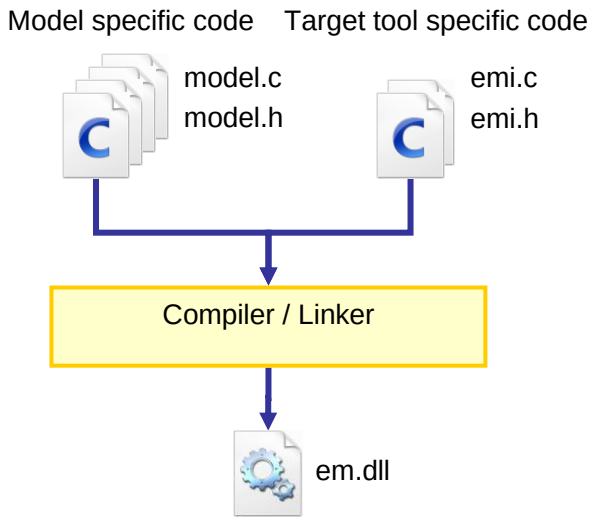


Figure 5: Work flow for hand-written external models

5.2 Tool-Generated External Models

For commercial CAE tools the automatic generation of external models is feasible. The Code Export Wizard integrated in SimulationX is already able to generate source code for various target platforms.

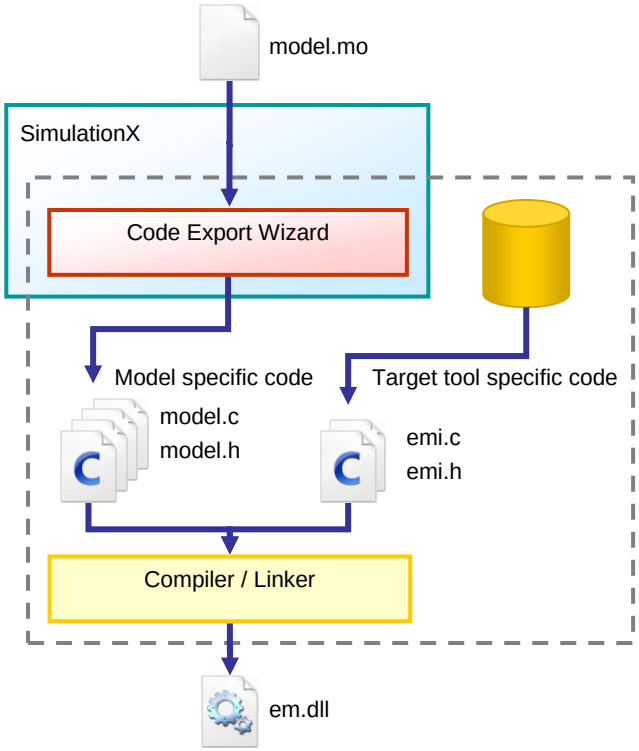


Figure 6: Work flow for tool-generated external models

Among S-functions for MATLAB/Simulink and UFORCE-routines for SIMPACK [5] also real time targets like ProSys-RT from Cosateq [6] are supported. For the automatic generation of EMI-conform model code a new target project type was added to the SimulationX Code Export Wizard. The wizard assists the user in the selection of inputs, outputs, and parameters. If a supported compiler is installed, SimulationX is able to build the External Model DLL immediately. The resulting model library does not need any additional runtime modules and can be distributed without limitations.

6 Conclusions and Outlook

We have shown how the interface to an external model in SimulationX is structured.

In one of the next Modelica Design meetings, we will make a proposal for the new predefined partial class `ExternalModel` which represents the model context inside the Modelica language.

The external model interface will be open for other software vendors. The interface itself does not contain Modelica specific parts. In this way external model components could be created and used by non-Modelica simulators too.

The authors explicitly invite interested colleagues for discussions about the interface proposal. A detailed specification is available on requested.

7 References

[1] http://www.simulationx.com

[2] http://www.gtisoft.com

[3] Modelica Association: Modelica A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version3.0, September 5th 2007.

[4] The MathWorks: Writing S-Functions (Manual), 2002.

[5] http://www.simpack.com

[6] http://www.cosateq.de