Implementation of Modelisar Functional Mock-up Interfaces in SimulationX

Christian Noll, Torsten Blochwitz, Thomas Neidhold, Christian Kehrer
ITI GmbH
Webergasse 1, 01067 Dresden, Germany
noll@iti.de, blochwitz@iti.de, neidhold@iti.de, kehrer@iti.de

Abstract

This document describes the implementation of the Modelisar Functional Mock-up Interfaces (FMI) in SimulationX. It presents the code generation of Functional Mock-up Units (FMU) for Model Exchange and Co-Simulation as well as the import of an FMU into SimulationX.

Keywords: Simulation; Modelisar; Functional Mock-up Unit (FMU); Functional Mock-up Interface (FMI)

1 Introduction

FMI stands for "*Functional Mock-up Interface*" [1] and was specified in the ITEA2 **Modelisar** project [2]. The intention is that dynamic system models of different software systems can be used together for software/model/hardware-in-the-loop simulation and for embedded systems. Using SimulationX Code Export, the functionality of a complete simulation model can be transformed into an FMU (*Functional Mock-up Unit*), which implements the FMI (*Functional Mock-up Interface*). A so created FMU can be instantiated by SimulationX or another simulation tool and accessed via the FMI functions. An FMU may either be self-integrating (*Co-Simulation*) or require the simulator to perform the numerical integration (*Model Exchange*).

2 FMU Support in SimulationX

There are two different FMI specifications (see Figure 1**:** FMI specifications), *FMI for Model Exchange* and *FMI for Co-Simulation*. Both are supported by SimulationX.

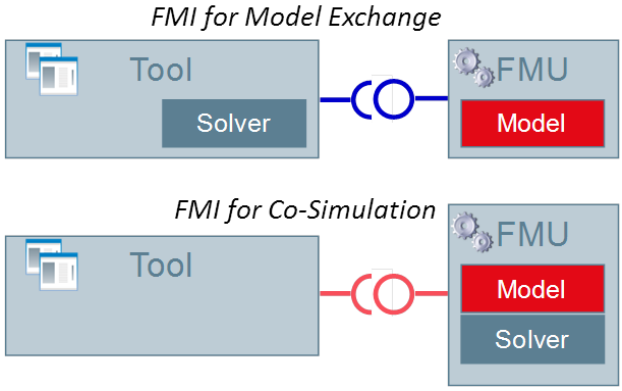

Figure 1: FMI specifications

2.1 FMU Code Export

Using SimulationX Code Export, the functionality of a complete simulation model can be transferred into an FMU (*Functional Mock-up Unit*). An FMU is distributed in the form of a zip File (*.fmu) and consists basically of the following components:

1. Exported Model + Interface
   The exported model functionality is accessible through standardized C-functions (FMI). By using the programming language C high portability is guaranteed. This component can be present as pure source code or as a binary (DLL). The FMI-Interface includes:
   - Functions for instantiation, initialization, termination and destruction.
   - Support of Real, Integer, Boolean and String inputs, outputs and parameters.
   - *Set* and *Get* functions for each type, e.g. *fmiSetReal(...)*.
   - Functions for exchange of simulation data, e.g. *fmiGetDerivatives(...)*

There is no explicit function call for the computation of the model algorithm. The FMU decides on its own, depending on which data have been set and the data being sought, which calculation is initiated. For efficiency it is important that variables are not newly computed, if they have been computed already at an earlier step. Instead they shall be reused. This feature is called “caching of variables” in the sequel.

representing a model which may then be easily integrated into another simulation environment.

The following illustration (see Figure 2: FMI Code Export for Model Exchange) shows the schematic workflow for transferring a SimulationX model into an *FMU for Model Exchange*.

After all desired inputs, outputs and parameters have been defined by the user in the Code Export Wizard, the code export process starts.

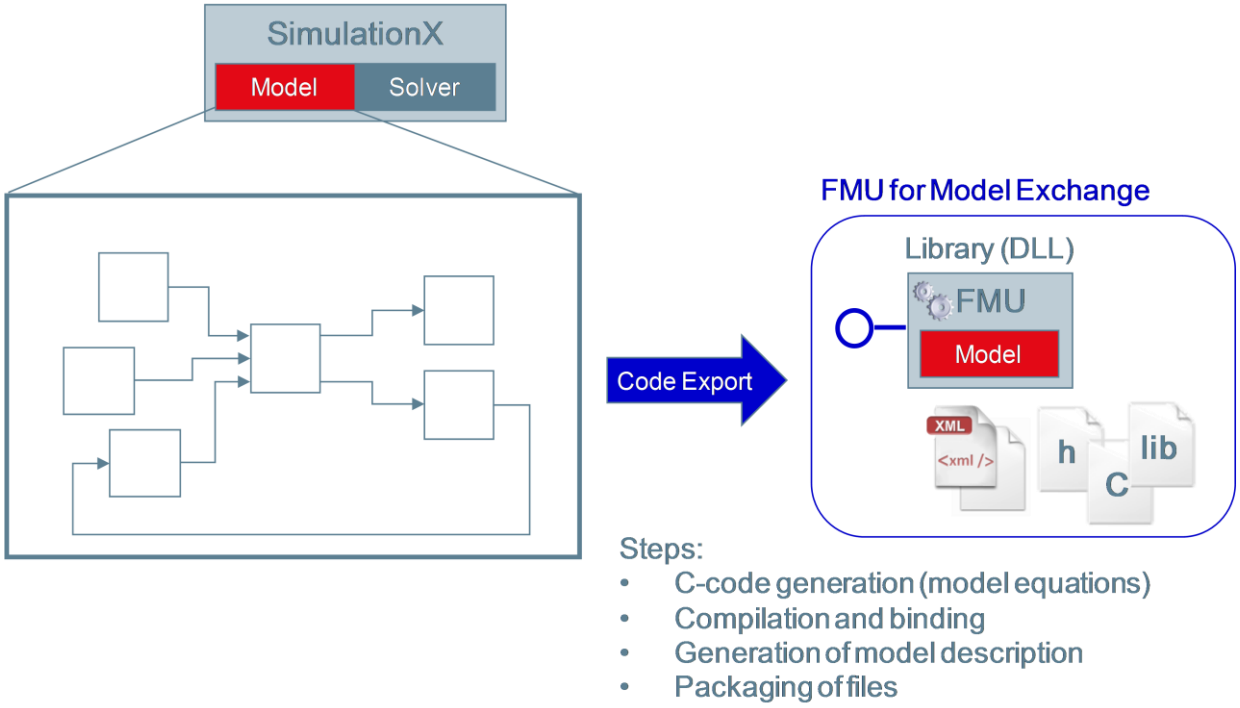

Figure 2: FMI Code Export for Model Exchange

2. Model Description Scheme
   This scheme is represented by an XML file that contains the description of the required data for the information flow between the FMU and the simulation tool. Through the description of the model within an XML file, the provider of simulation tools are not forced to use a specific representation of data structures.
3. Data and Documentation (optional)
   Additional data and documentation of the model can be included.

During the code export the following steps are executed. At first a special symbolic analysis will transfer the model into ordinary differential equations. Based on this equations and the defined interface, the C-code that includes the model functionality and the specific FMI interface functions, is generated. Furthermore the XML model description file is generated. At the end of this process a zip-file (*.fmu), with all necessary files, is created to distribute the FMU.

2.1.1 FMI for Model Exchange

The intention of *FMI for Model Exchange* is to allow any modeling tool to generate C code or binaries

2.1.2 FMI for Co-Simulation

The FMI for Co-Simulation is an interface standard for the solution of time dependent coupled systems consisting of subsystems that are continuous in time

(model components that are described by differential equations) or time-discrete (model components that are described by difference equations like, e.g. discrete controllers).

The FMI for Co-Simulation defines interface routines for the communication between a master and the individual simulation tools (*slaves*) in the co-simulation environment. The data exchange is restricted to discrete communication points in time and the subsystems are solved independently between these communication points.

The following illustration (see Figure 3**:** FMI Code Export for Co-Simulation) shows the schematic workflow to transfer a SimulationX model into an *FMU for Co-Simulation*.

simulation environment, is provided. In particular, this includes a set of capability flags to characterize the ability of the slave to support advanced master algorithms. One of these flags is *canHandleVariableCommunicationStepSize* that specifies whether the slave can handle variable communication step size. Another flag is *canRejectSteps* that indicates the slave's capability to discard and repeat a communication step. This will be supported in a future SimulationX release.

The flag *canInterpolateInputs* defines that the slave is able to interpolate continuous inputs. In this case, calling of *fmiSetRealInputDerivatives(...)* has an effect for the slave. At the end of the export process a zip-file (*.fmu) is created to distribute the FMU.

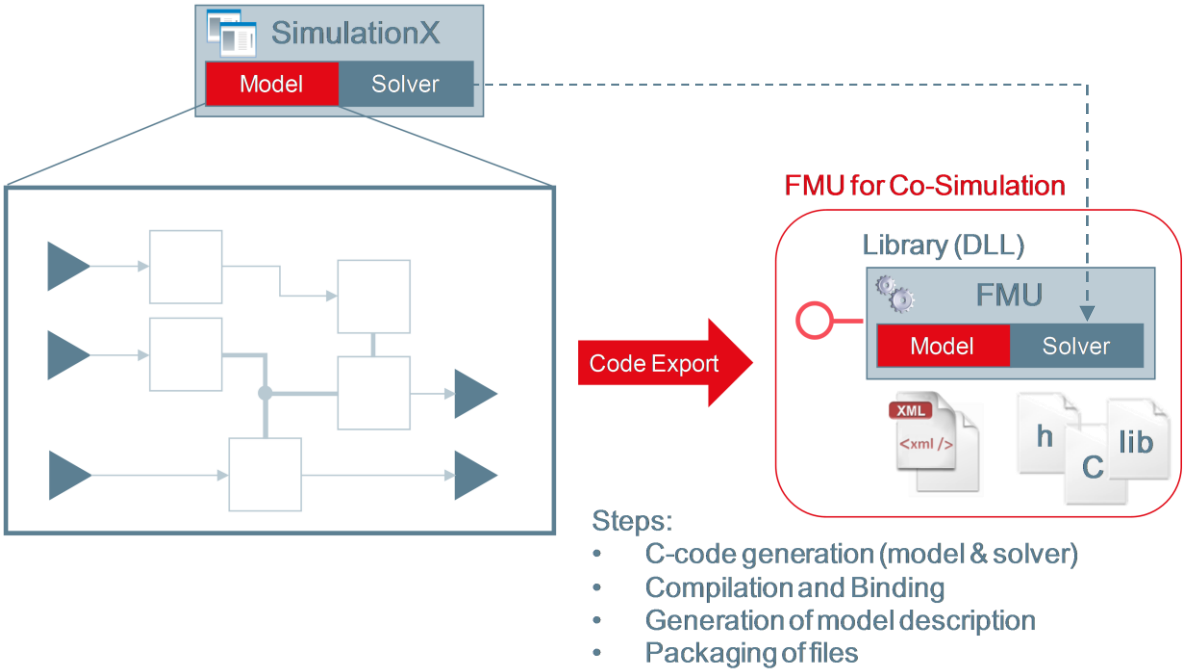

Figure 3: FMI Code Export for Co-Simulation

After all desired inputs, outputs and parameters have been defined by the user in the Code Export Wizard the code export process starts. During the Code Export the following steps are executed: At first a special symbolic analysis will transfer the model into ordinary differential equations. Based on this equations and the defined interface, the C code, that includes the model functionality, the specific FMI interface functions and a Solver (CVODE), is generated. The Sundials CVODE solver [4] uses a BDF variant and is well suited for stiff models.

Furthermore the XML model description file is generated, where all information about the slaves, which is relevant for the communication in the co-

2.2 FMU Import

The SimulationX FMU import consists of unzipping the *.fmu file and the generation of Modelica code including the calls of FMI functions based on the XML model description. A re-export via code export is supported.
The main idea of embedding a FMU into a Modelica model is to construct an external object and some external functions to interact with that model.

The automatic import process is started by selecting the menu ***Insert/Functional Mockup Unit….***
Thereupon the following dialog (see Figure 4**:** FMU Import Dialog) for importing a FMU appears.

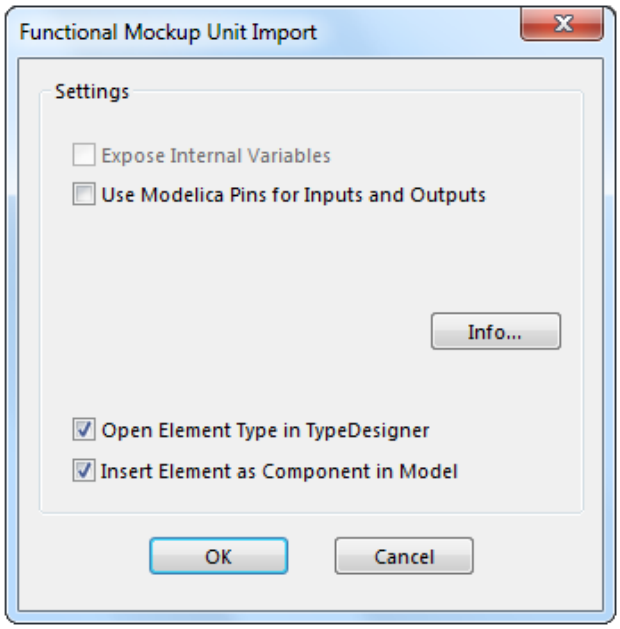

Figure 4: FMU Import Dialog

During the import process the DLL and Lib files (if any exist) are copied to the External Function folder.

2.3 Tool Coupling

The current version 1.0 of the FMI for Co-Simulation standard not only allows the coupling of specially prepared software modules (FMU), but can also be used for direct coupling of CAE tools. Thereto the particular application with its proprietary interfaces is made available via a special wrapper (see Figure 5: Tool coupling via wrapper DLL) that implements the standardized Functional Mock-up Interface and provides it for other applications. From the outside, the particular application behaves like a Functional Mock-up Unit.

For SimulationX such a wrapper will be available. The implementation is based on the existing COM interface of SimulationX. For integrating a Modelica model in such a co-simulation an adequate preparation is necessary. Especially the inputs, outputs and parameters of the model have to be defined. All this information is stored as a "real FMU" in a zip archive. The model itself or a link to this model in the local file system or on the network must also be stored in this file.

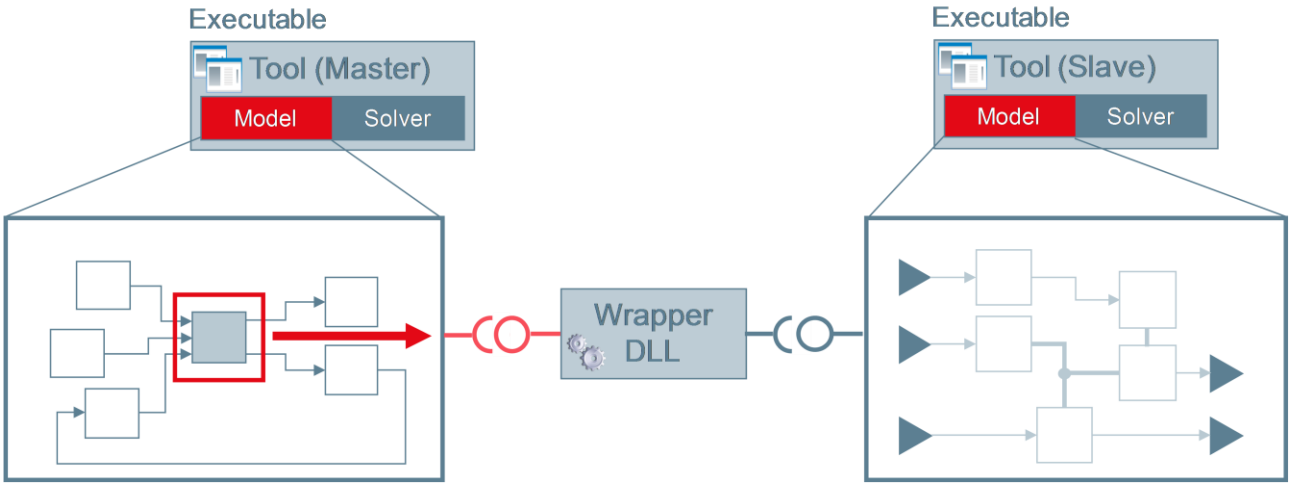

Figure 5: Tool coupling via wrapper DLL

To link an FMU with a Modelica model SimulationX uses an *External Object*. The *fmiInstantiateModel(...)/ fmiInstantiateSlave(...)* and *fmiFreeModelInstance(...)/ fmiFreeSlaveInstance(...)* functions are called as constructor and destructor, respectively. All other functions are called as external functions with an external object as first parameter. Because the *fmiInitialize(...)* function has to forward function pointers for several purposes, this function is redirected through a special built-in function.

3 Implementation Difficulties

During the implementation of the FMU import into SimulationX as a Modelica simulator a few difficulties had to be overcome.

The first problem is related to a type mismatch between *fmiBoolean* and Modelica *Boolean*, which leads to a type cast for scalar *fmiBoolean* variables and the necessity of restoring *fmiBoolean* arrays.

Further, there is an issue concerning the *fmiInstantiateModel* function, because the argument *fmiCallbackFunctions* is a *struct* that holds function pointers. In Modelica there is no possibility to generate such a record.

Also, it is not easy to implement that the function *fmiInitialize* is called only once, because according to Modelica language specification the body of a *when initial()* clause may be traversed several times during initialization.

Changing of discrete variables is only allowed in Modelica at event steps, not during continuous integration, but the *fmiSetXxx* functions returns *fmiStatus* as a Modelica integer variable, which is a discrete variable. Hence a Modelica compiler may call such functions only at event time instances. But the *fmiSetXxx* functions have to be called during continuous integration too.

The functions *fmiCompletedIntegratorStep*, *fmiEventUpdate*, and *fmiTerminate* are impure and thus may not be treated like constant functions. But, there is no possibility in Modelica to mark a function as impure.

There are two difficulties related to the FMI calling sequence. First, there is no possibility in Modelica to be informed about the reason for a model computation. But, it is relevant to distinguish between calling for instance *fmiEventUpdate* or *fmiCompletedIntegrationStep*. Secondly, Modelica does not provide the functionality to trigger an event step and call *fmiEventUpdate*.

Modelica has no functionality to provide event indicators (*evi*) directly. According to FMI specification FMUs have to add a small hysteresis to the *evi*. A Modelica tool may do the same with its internal root functions. Hence the hysteresis is added twice and events caused by the FMU are located a little bit inaccurately.

We solved these problems by using some internal Modelica extensions in SimulationX, which we also proposed to the Modelica Language Design Group and accordingly to the FMI standard committee.

4 Conclusions

With the Modelisar FMI standard exists a vendor-neutral interface that allows the exchange of simulation models between different tools and platforms. The chances to establish FMI as a standard are pretty good, because software vendors and users were involved right from the start. At the end, the success of this interface is measured by how the tool vendors will integrate FMI into their products. In addition to reliability and numerical stability the ease of use will determine this success.

References

[1] Functional Mock-up Interface: http://www.functional-mockup-interface.org/index.html

[2] ITEA2 Modelisar Project: http://www.modelisar.com

[3] Arnold M., Blochwitz T., Clauß C., Neidhold T., Schierz T., Wolf S., FMI for Co-Simulation: Multiphysics Simulation - Advanced Methods for Industrial Engineering. Bonn, June 2010.

[4] M. Otter, T. Blochwitz, H. Elmqvist, A. Junghanns, J. Mauss, H. Olsson: Das Functional Mockup Interface zum Austausch Dynamischer Modelle. Plenary talk at the ASIM workshop. Ulm, 4. - 5. March 2010.

[5] Neidhold T. Tool Independent Model Exchange Based on Modelisar FMI. Industrietag Informationstechnologie. Halle(Saale), May 2010.

[6] SUNDIALS: https://computation.llnl.gov/casc/sundials/main.html.

[7] FMI for Model Exchange v1.0: http://www.functional-mockup-interface.org/specifications/FMI_for_ModelExchange_v1.0.pdf

[8] FMI for Co-Simulation v1.0: http://www.functional-mockup-interface.org/specifications/FMI_for_CoSimulation_v1.0.pdf