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