Master for Co-Simulation Using FMI

Jens Bastian Christoph Clauß Susann Wolf Peter Schneider
Fraunhofer Institute for Integrated Circuits IIS / Design Automation Division EAS
Zeunerstraße 38, 01069 Dresden, Germany
{Jens.Bastian, Christoph.Clauss, Susann.Wolf, Peter.Schneider}@eas.iis.fraunhofer.de

Abstract

Co-Simulation is a general approach to simulate coupled technical systems. In a master-slave concept the slaves simulate sub-problems whereas the master is responsible for both coordinating the overall simulation as well as transferring data. To unify the interface between master and slave the FMI for Co-Simulation was developed. Using FMI a master was implemented with simple and advanced algorithms which can be applied depending on the properties of the involved slave simulators. The master was tested amongst others by coupling with SimulationX.

Keywords: co-simulation; FMI; master

1 Introduction

Modeling problems in natural sciences and engineering often leads to hybrid systems of differential and algebraic, time continuous and time or event discrete equations. Often complex multi-disciplinary systems cannot be modeled and simulated in one simulation tool alone or subsystem models are available only for a specific simulation tool. Sometimes sub-problems shall be simulated with the simulator which suits best for the specific domain. Thus for the simulation of multi-disciplinary problems or for hardware-in-the-loop simulation it is often reasonable or even necessary to couple different simulation tools with each other or with real world system components.

Simulator coupling is used in various fields of application like automotive engineering, microelectronics, mechatronics etc.

Up to now simulator coupling is nearly always a point-to-point solution tailored to the involved simulators. These special solutions cause high effort so a generally accepted interface for simulator coupling supported by many simulation tools is desirable.

2 Co-Simulation

Co-simulation is an approach for the joint simulation of models developed with different tools (tool coupling) where each tool treats one part of a modular coupled problem. Intermediate results (variables, status information) are exchanged between these tools during simulation where data exchange is restricted to discrete communication points. Between these communication points the subsystems are solved independently.

2.1 Coupling of simulators

A simulation tool *S* can be coupled if it is able to communicate data during simulation at certain time points *t*, cf. Figure 1. Here input variables are denoted by *u* and output variables by *y*.

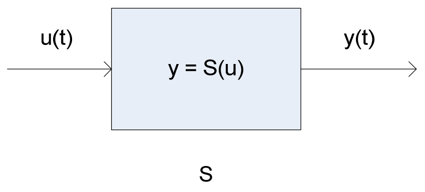

Figure 1: Block representation of a simulator S

Simulators have different capabilities which have an influence on the algorithms that can be used for their coupling. Such capabilities are:

- The simulator can handle variable communication step sizes.
- The simulator can handle events.
- It is possible to undo a time step, i.e. the simulator can reject time steps.

When using simulator coupling the original problem is divided into *N* subproblems each handled by a simulator. Typically, *N* is small, i.e. below 20. Thereby the simulators do not have to be different.

The signal flow for the coupled simulators can be described by a directed graph with the simulators as the nodes and the exchanged data as the edges.

If there is feedback in the graph then cycles exist. A cycle is a path in a graph with the same node as start and end point. Cycles can be eliminated if the simulators in a cycle are combined into a super-simulator i.e. a simulator superior to the simulators of the cycle.

Figure 2 shows an example of such a graph. Simulator A has the highest priority. The simulators B, C, and D form a cycle. E, F, and G are subordinated to this cycle. That means simulator A is executed first of all. Then the cycle of B, C, and D is finished. Afterward simulators E and F are executed whereat both simulators can be run in parallel. At last G is processed.

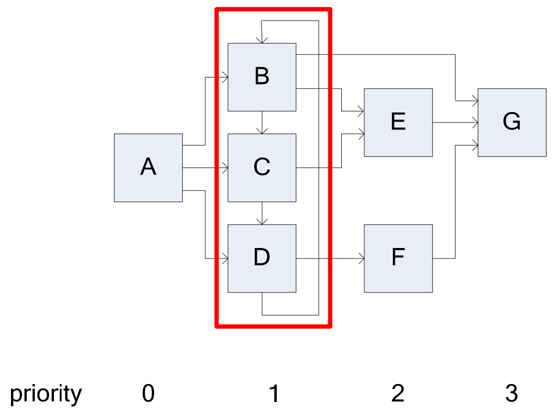

Figure 2: Example graph of coupled simulation tools

For simulation, the whole graph is analyzed first. If cycles are detected then they are combined into a super simulator. The simulators are coupled with directed data flow. A priority is assigned to each simulator with 0 representing the highest priority. Simulators with the same priority can be executed in parallel. All simulators in cycles either have to be processed iteratively or with small enough time steps and error control.

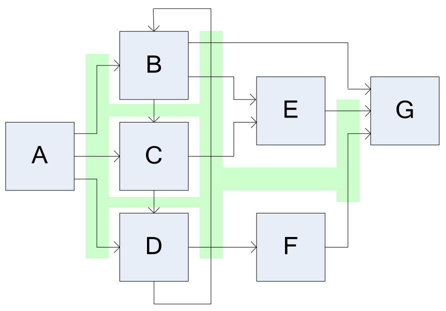

Figure 3: Master-Slave structure

Instead of direct coupling, a master is assumed to be located between the single simulation tools which synchronizes, controlles and manages them [1]. Each edge of the graph is regarded as to go “through” the master, cf. Figure 3. The master serves as an interface, establishes connections and exchanges data between the simulators which are called slaves. Slaves are assumed to communicate with the master only.

2.2 Basic Co-simulation computational flow

The whole co-simulation can be divided into several phases.

1. Initialization phase

All simulation tools are prepared for starting the co-simulation. The master receives the properties of the slaves. Furthermore the master receives the connection graph. The slaves and models are initialized and parameters are set. The communication links between master and slaves are established. The master chooses its algorithm based on the capabilities of the slaves as well as the connection graph and user input.

2. Simulation phase

The master forces the slaves to simulate the time interval from start time to stop time by stepwise solving master subintervals which are also called communication steps. Their boundaries are called communication points. In case of event iteration the communication step size can be zero. The simulation is performed independently for all subsystems restricting data exchange between subsystems to these communication points.

Before simulating a subinterval a slave receives its input values and possibly their derivatives with respect to time as well as the communication step size from the master. After finishing the communication step the master receives the output values of the slave and possibly their derivatives with respect to time. Furthermore the slave status has to be transferred to the master. If the slave simulation fails further communication is necessary.

3. Closing phase

The master stops the complete simulation and is responsible for proper memory deallocation, terminating and resetting or shutting down the slaves.

2.3 Accuracy and stability

Co-simulation can lead to problems regarding stability and accuracy of the simulation [2] – especially if feedback exists between simulators, cf. the example given in section 4.4. If a simulation tools provides an interface for co-simulation at all then usually it is not possible to reset a simulator so that a time step can be repeated e.g. with a smaller step size.

So co-simulation should often be used as a last resort as long as iterative methods have to be used for stability and simulators only provide a rudimental co-simulation interface. Hopefully this will change in future with the introduction of a standardized co-simulation interface like the one proposed in the next section.

3 Functional Mock-up Interface (FMI) for Co-Simulation

The Functional Mock-up Interface (FMI) for Co-Simulation [3], [4], [5] is an interface standard for the solution of time dependent coupled systems consisting of subsystems that are continuous in time or time-discrete. It provides interfaces between master and slaves and addresses both data exchange and algorithmic issues. Both simple as well as more sophisticated master algorithms are supported. However, the master algorithm itself is not part of FMI for Co-Simulation.

FMI for Co-Simulation consists of two parts:

- *Co-Simulation Interface*: a set of C functions for controlling the slaves and for data exchange of input and output values as well as status information.
- *Co-Simulation Description Schema*: defines the structure and content of an XML file. This slave specific XML file contains "static" information about the model (input and output variables, parameters, …) and the solver/simulator (capabilities, …).

The complete interface description can be obtained from [3].

The capability flags in the XML file characterize the ability of the slave to support advanced master algorithms which use variable communication step sizes, higher order signal extrapolation etc.

A component implementing the FMI is called Functional Mock-up Unit (FMU). It consists of one zip file containing

- the XML description file and
- the implementation in source or binary form (dynamic library).

A master can import an FMU by first reading the model description XML file contained in the zip file.

Coupling simulators by FMI for Co-Simulation hides their implementation details and thus can protect intellectual property.

FMI for Co-Simulation version 1.0 was published in October 2010. Currently it is planned to combine FMI for Co-Simulation with FMI for Model Exchange to an FMI standard.

4 EAS Master

MODELISAR [6] is a research project within the European ITEA2 program. It is aimed to develop the FMI as well as to support it by involved tool vendors. Use cases will show the benefits of applied FMI. Master algorithms are not standardized with FMI but developed in the MODELISAR project e.g. by tool vendors. A prototypical implementation of a master has been provided by EAS for the MODELISAR consortium. The package contains the ANSI C code of the master, a generic "C function" slave, and a collection of examples.

The "C function" slave provides the basic functionality of FMI for Co-Simulation. The user has only to provide two functions for initialization (the number of input and output variables) and the computation of a step with the step size communicated by the master.

4.1 Configuration

The master is configured by a simple text file. There are keywords for start and stop time, step size, coupling algorithm, error tolerance etc. The coupled FMUs with their paths have to appear within the configuration file, too. The graph of the simulator coupling has to be supplied by an incidence matrix and information about the priority of the slaves as well as occurring cycles.

4.2 Coupling algorithms

The master prototype provides three algorithms for the simulation with fixed step size:

- data flow between the slaves without iterations, i.e. simple forward calculation
- fixed point iteration of all cycles within the graph
- simple implementation of Newton's method with Jacobians approximated by finite differences

All master algorithms proceed in macro steps of fixed step size from start time to end time.

The computation of a time step from t_i to t_{i+1} within cycles is performed in the following way: Every slave makes an assumption for its input value u at time t_{i+1}. Currently this is done using constant interpolation $u(t_{i+1}) = u(t_i)$, i.e. in each macro step all terms that couple the subsystems are frozen. Thus synchronization and update of the exchanged values with computed output $y(t_{i+1})$ is done at the end of the time step. Because no slave depends on the current output of another one, the slaves can run in parallel. This iteration scheme is called to be of Jacobi type.

Another approach would be to simulate a time step with every slave of a cycle one after another and to use the output $y(t_{i+1})$ just calculated as input $u(t_{i+1})$ for the following slaves. These staggered algorithms which handle the subsystems sequentially are called of Gauß-Seidel type. This method was used within a first master implementation. The drawback of this approach is that the slaves within the cycles cannot run in parallel and the behavior of the iteration depends on the calling sequence of the slaves. However, an example exists where this approach converges while the first method does not converge.

4.3 Simple slave test examples

A collection of examples using the "C function" slave is provided together with the master. They cover different types of coupling – with or without cycles, nonlinear equations, ODEs, DAEs – and demonstrate the usage of the configuration file. Some of the examples can be solved with all master algorithms, some only with Newton's method.

One of these examples is BspK6. It consists of four coupled slaves which exchange 4 values (0, 1, 2, 3) of type `fmiReal` and 2 values (4, 5) of type `fmiInteger`, cf. Figure 4. The slaves S0, S1, and S2 form a cycle.

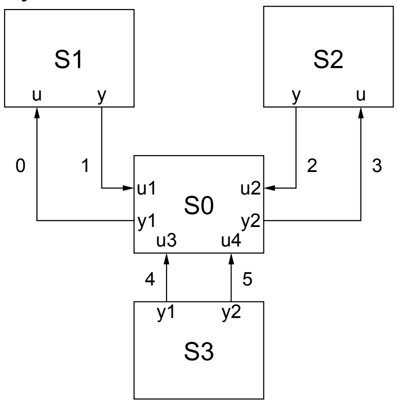

Figure 4: Example BspK6 from collection

Input, output, and internal variables of the slaves are related by the following equations.

Slave S0:

$$\frac{\mathrm{d}x}{\mathrm{d}t} + 2x - u_1 - u_2 = 0$$

$$y_1 := \begin{cases} 0 & u_4 = 1 \\ u_1 - x & \text{else} \end{cases}$$

$$y_2 := \begin{cases} 0 & u_3 = 1 \\ u_2 - x & \text{else} \end{cases}$$

Slave S1:

$$\frac{1}{2}\frac{\mathrm{d}x}{\mathrm{d}t} + x + u - \sin(3\pi t) = 0$$

$$(y - x)/\left(1000\sin\left(\tfrac{2}{10}\pi t\right) + 1001\right) - u = 0$$

Slave S2:

$$\frac{1}{2}\frac{\mathrm{d}x}{\mathrm{d}t} + x + u - \sin(2\pi t) = 0$$

$$(y - x)/\left(-1000\sin\left(\tfrac{2}{10}\pi t\right) + 1001\right) - u = 0$$

Slave S3:

$$y_1 = \begin{cases} 1 & \sin(\pi t) > \frac{1}{2} \\ 0 & \text{else} \end{cases}$$

$$y_2 = \begin{cases} 1 & \sin(2\pi t) < -\frac{1}{2} \\ 0 & \text{else} \end{cases}$$

Figure 5 and Figure 6 show simulation results for constant step size 10^{-4} and Newton's method as iterative method for the cycle. The other two methods do not converge for this example.

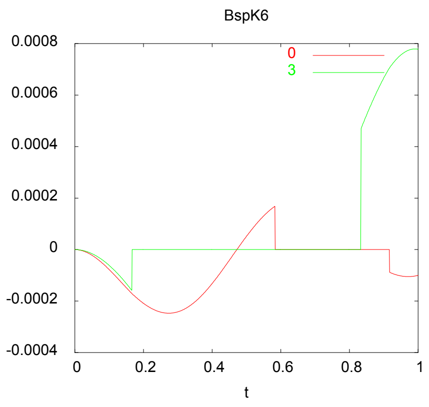

Figure 5: Simulation results for exchanged values 0 and 3

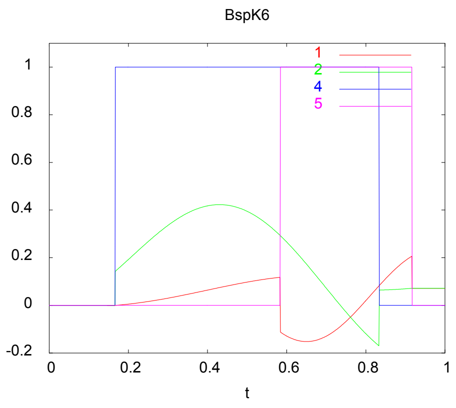

Figure 6: Simulation results for exchanged values 1, 2, 4, and 5

4.4 Coupling with ITI SimulationX

Another example shows coupling of SimulationX [7] with a "C function" slave via the EAS Master.

The original SimulationX model is shown in Figure 7. It is a simple plant with a controller driven by the "speed" function

$$f(t) = \begin{cases} 100\,\text{rpm} & 0.2\,\text{s} < t < 1\,\text{s} \\ 0 & \text{else} \end{cases}$$

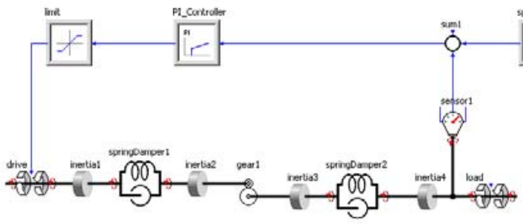

Figure 7: Full SimulationX model

This model has been split into three FMUs: two SimulationX FMUs for the controller and the plant and one "C function simulator" FMU for the speed input, cf. Figure 8. The SimulationX FMUs contain the model as well as the solver as a DLL. They were created via the code export option of SimulationX.

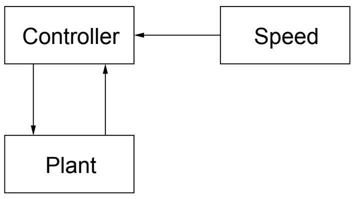

Figure 8: Coupling of three FMUs

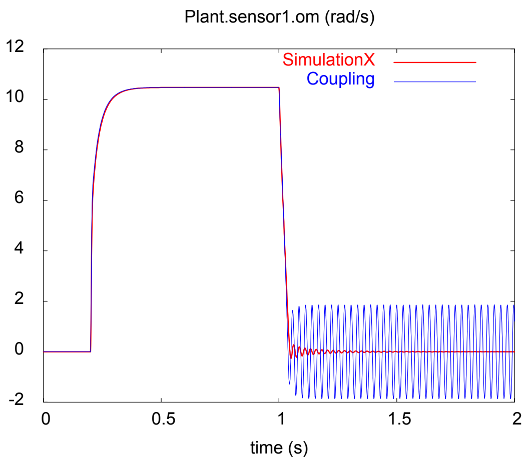

Figure 9: Simulation results

The coupled FMUs have been simulated by the prototypical master with fixed step size 10^{-3} with the simple algorithm for forward calculation without iteration. Results of this calculation as well as of the original simulation model are presented in Figure 9.

As it can be seen, the angular velocity of the plant shows a small but fast decaying oscillation in the original model after the speed has been switched to 0 after 1 s. In contrast, the oscillation is larger and does not decay in the simulator coupling. For larger step sizes the amplitude of this oscillation is even larger (not shown).

At the moment, SimulationX cannot discard steps so a simulation with iterative methods was not possible. With iterative methods we expect the oscillation to decay like in the original model.

4.5 Efficiency

Efficiency and simulation speed strongly depend on the problem which has to be solved.

Clearly, the most efficient approach would be to use only one simulation tool and do without co-simulation. If this is not possible then problems described by graphs without feedback can be simulated most efficiently using the non-iterative method. If there are cycles within the graph and no iterative methods can be used because the simulators cannot discard steps then accuracy and numerical stability may be poor. Anyway, the macro step size has to be very small then and thus the computational costs strongly increase.

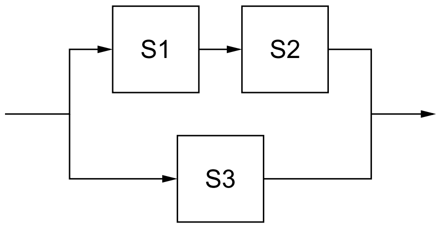

Figure 10: Disadvantage of current OpenMP approach compared to thread programming

By using OpenMP [8] slaves of the same priority can run in parallel. However, the current implementation of this approach has a disadvantage compared to thread programming which will be explained with the help of Figure 10. Here S1 has a higher priority than S2. S3 can have the same priority either as S1 or S2. Thus either both S3 and S1 can run in parallel and the simulation continues with S2 after both S1 and S3 have finished or S2 and S3 can both run in parallel after S1 has finished. Instead it would be better to handle S1 and S2 as a "super slave" which runs in parallel with S3, i.e. synchronization takes place at the start of S1 and S3 and after S2 and S3 have finished. However, either a more complicated data structure has to be used for this purpose if OpenMP should be used or platform dependent thread programming has to be used.

4.6 Summary of properties

The implementation of the EAS Master is as platform independent as possible. Platform dependent code – mainly for dealing with dynamic libraries – could happily be collected as preprocessor defines within a single header file. Thus the master runs on multiple platforms (MS Windows, Linux, Sun Solaris).

Slaves can run in parallel if they have the same priority. Platform independence also was the reason to use OpenMP instead of explicitly dealing with thread programming for this purpose. OpenMP is supported by newer version of the major C compilers (gcc, Visual Studio). Parallelization is realized by one #pragma directive in front of a "for" loop so that compilers without OpenMP support simply compile the code for serial execution. However, the OpenMP approach has the drawback compared to explicit dealing with threads that only slaves of the same priority and not across different priorities can run in parallel.

Currently the three algorithms mentioned in section 4.2 are available.

4.7 Future enhancements

A commercially available version of the master will have the following features:

- The graph will automatically be analyzed for the priority of the slaves and cycles.
- Newton's method will be improved. A better Jacobian update strategy will be used so that the high cost of calculating a new Jacobian by finite differences will be reduced.
- Broyden's method will be available as another iterative method.
- A step size control will be implemented based on results in [9] so that variable macro steps can be used.
- Polynomial interpolation of data besides the currently used constant interpolation will be supported.

5 Conclusions

Co-simulation is a powerful method to simulate heterogeneous systems where each subsystem is simulated by its own specialized simulator. However, currently simulation tools have their own interface for coupling – if at all. Additionally, they are often not able to discard steps and thus not suitable for iterative methods.

The Functional Mock-up Interface (FMI) for Co-Simulation as a proposed standard for simulator coupling will hopefully be widely used because it replaces current point-to-point solutions and thus eases the reuse of models tailored to special simulators. The protection of intellectual property is also possible with FMI.

Providing the prototypical master implementation will hopefully help to promote the FMI for Co-simulation.

Acknowledgements

The SimulationX model and FMUs were kindly provided by T. Blochwitz from ITI.

This work is supported by the German BMBF within the ITEA2 MODELISAR project.

The authors thank the reviewers for valuable remarks.

References

[1] Wolf, S.; Blochwitz, T.: Master Slave Simulator Coupling. ITI Symposium 2010.

[2] Schierz, T.; Arnold, M.: Advanced numerical methods for co-simulation algorithms in vehicle system dynamics. 1st Conference on Multiphysics Simulation, Bonn 2010.

[3] Functional Mock-up Interface for Co-Simulation v1.0, MODELISAR consortium, 2010. http://functional-mockup-interface.org

[4] Arnold, M.; Blochwitz, T.; Clauß, C.; Neidhold, T.; Schierz, T.; Wolf, S.: FMI-for-CoSimulation. 1st Conference on Multiphysics Simulation, Bonn, 2010.

[5] Enge-Rosenblatt, O., Clauß, C.; Schneider, A.; Schneider, P.: Functional Digital Mock-up and the Functional Mock-up Interface – Two Complementary Approaches for a Comprehensive Investigation of Heterogeneous Systems. 8th International Modelica Conference, Dresden, 2011.

[6] http://www.modelisar.org

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

[8] http://www.openmp.org

[9] Schierz, T.; Arnold, M.; Eichberger, A.; Friedrich, M.: Study on Theoretical and Practical Aspects of Communication Stepsize Control. MODELISAR, sWP203 report, 2010.