A CASE STUDY ON FMU AS CO-SIMULATION EXCHANGE FORMAT FOR FEM MODELS

Christof Kaufmann and Jörg Frochte*
Dept. of Electrical Engineering and Computer Science
Bochum University of Applied Science
42579 Heiligenhaus, Germany
{christof.kaufmann, joerg.frochte}@hs-bochum.de

ABSTRACT

This work deals with the *Functional Mock-Up Interface*, which is becoming more and more popular in simulation tools since 2011. Developed with a focus on ordinary differential equation or differential algebraic equation based models arising from Modelica models we discuss the extension on Partial Differential Equation based models discretized using the finite element method (FEM). The suggested approach allows to generate *Functional Mock-up Units* (FMU) for co-simulation purpose without the need of classical tool coupling and thus without all its draw-backs concerning interoperability and intellectual property protection. For the suggested open source components – mainly the SUNDIALS package for time integration and the UMFPACK solver for sparse linear systems – a case study was performed concerning aspects of integration and efficiency including numerical issues.

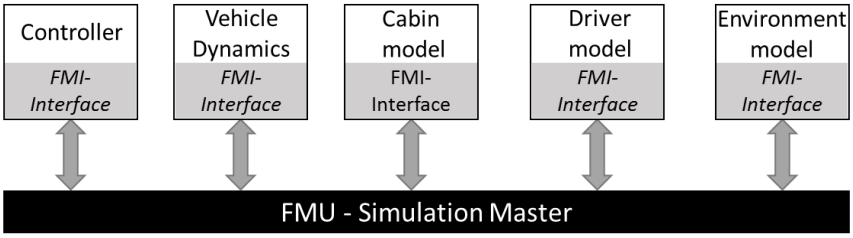
KEYWORDS

FMU/FMI, Co-Simulation, FEM, Modelica, Distributed Computing, Numerical Software, Open Source

1. INTRODUCTION

Nowadays, simulation is used in many areas in industry. Interoperability between different simulation areas and specialized tools becomes more and more important. Next to interoperability, intellectual property protection for modelling and simulation know-how is an issue for some industrial users, especially in complex supply scenarios that are typical e. g. for the automotive industry. To solve this problem set, academic and industrial partners – Daimler and Dassault among others – formed 2011 in the ITEA 2 European project *MODELISAR* a software interface to either export models or let the tools interact with each other, see e. g. (Blochwitz, T., et al. 2011). This is called Functional Mock-Up Interface (FMI), see (Modelica Association, 2014). The idea is to enable to export a model in a defined way and make it accessible via an interface, which is supported by many tools. Such an exported model is called Functional Mock-up Unit (FMU).

Figure 1 Usage Scenario of FMI with a FEM model



Technically, an FMU is a shared library that implements the FMI. The FMI defines two kinds of FMUs to serve different purposes: *Model Exchange FMU* and *Co-Simulation FMU*. An important difference between these two kinds is that in Model Exchange (ME) the FMU is simulated using the importing tool's technology and solver, while in Co-Simulation (CS) the FMU contains its own solver. Using a co-simulation approach the

main purpose of the simulation master in Figure 1 is to organize the data exchange between the FMUs at discrete communication points in time. One drawback of the co-simulation approach is that there are additional issues concerning errors related to co-simulation as technique, see e. g. (Arnold, M., et al. 2014). Using a sufficiently intelligent back-end as simulation master the model exchange FMU could prevent these effects.

Nevertheless, we will focus in this work on the co-simulation approach. The reason is that the common back-ends of tools supporting FMI are not sufficient for finite element method (FEM) simulation. The FMI project was mainly driven by the industrial and academic Modelica community. Modelica as modelling language, see e. g. (Peter Fritzson 2004), is based on differential algebraic equations (DAEs) as they arise from block connected systems or lumped models. One result is e. g. that FMI is focused towards scalar variables; there is no vector variable defined in the interface. Also, most time-integrators in tools providing an FMI import are focused on DAEs or ordinary differential equations (ODEs). However, often it would be desirable to simulate discretized partial differential equations (PDEs) with a geometry, i. e. spatial models, together with other DAEs. This is also the focus in the papers (Stavåke, K., et al. 2013), (Stavåke, K., et al. 2014). They use a framework for the FMU to include the whole workflow of an FEM simulation, including mesh generation. In this work, our proposed FMU framework focuses only on the last step of a finite element simulation, which is the numeric solving. This allows a lightweight and simple FMU framework. It is easy to write an extension for a tool to generate the necessary bits from the FEM model to complete the framework to gain a stand-alone FMU.

FEM is a popular method to discretize PDEs. Using the method of lines results in a large, sparse ODE or a DAE system. The unknown is a vector of large dimension, which does not suit directly to the scalar variables of the FMI. Also, the back-ends of today's tools that have the possibility to import an FMU, will suffer on the task of handling the simple structured but large systems from the FEM. These tools focus more on techniques like the ones described in (Cellier, F. E., & Kofman, E., 2006) and handling aspects like structural singularity, ordering equations etc. This is probably one of the reasons why approaches like the one presented in (Zheng 2008) are still not popular. Therefore the only practical way to exchange FEM Models that contain a few hundred thousand unknowns and transfer into a tool that implements the FMI is with a co-simulation FMU.

There are some more approaches importing FEM into Modelica based tools, with different drawbacks. In most cases one is bounded to a specific tool, see e. g. (Khan, S. 2012). Another possibility to embed FEM models in a CS FMU is to use FMI just as wrapper for a tool coupling approach. One drawback of shipping a model to another supplier or customer using this approach is that it requires all users to have a license of the specific FEM tool, which is not desirable concerning interoperability. And in most cases one needs to send an accessible model for the simulation, which contradicts intellectual property protection.

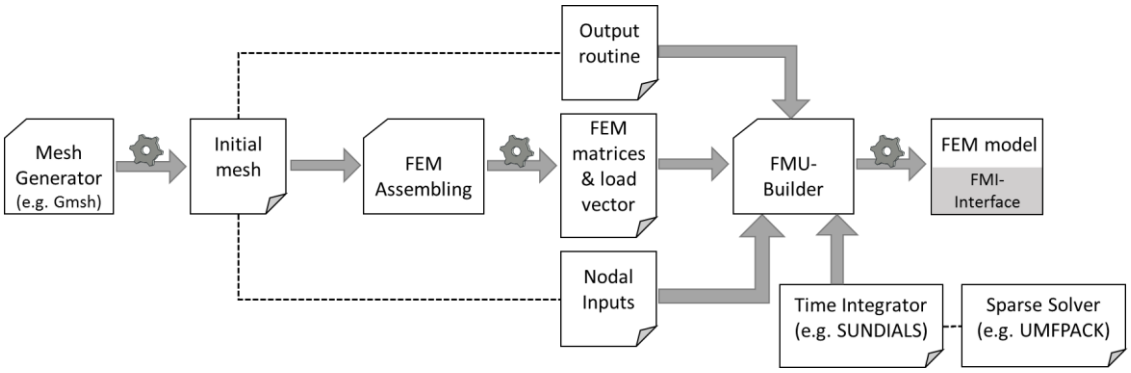
In this work we suggest a workflow to build CS FMUs based on open source components. Beyond this, we present a case study how it is possible to combine the FMI with well-proven open source software libraries, like the integrators from the *SUite of Nonlinear and DIfferential/ALgebraic Equation* (SUNDIALS) and the sparse solver UMFPACK, see e. g. (Davis T. A. 2004), or SuperLU, see e. g. (Xiaoye S. Li, 2005). The results are stand-alone FMUs that have the capability to be transferred without license restriction to every tool implementing the FMI standard. The remainder of this paper is organized as follows: Section 2 provides a lightweight framework for the generation of FEM FMUs, while section 3 discusses the issues concerning different approaches of time integrator techniques. In section 4 we present some benchmarks and results. The paper finishes with conclusions and future prospects.

2. FRAMEWORK FOR FMU GENERATION BASED ON OPEN SOURCE COMPONENTS

The suggested framework achieves two different goals. It is simple and lightweight on the one hand and the generated FMU is stand-alone solely using open source software without licensing issues on the other hand. For the lightweight FEM-FMU scenario shown in Figure 2 we consider a use case in which the spatial mesh is constant during the whole simulation. In this case the mesh is only generated once before simulation. Based on the mesh the assembling of the FEM matrices and the load vector is performed. In most FEM models the load vector contains source terms e. g. heat sources. Besides boundary conditions, source terms are the typical part where input values u from the simulation master in a co-simulation scenario are employed. The FEM-FMU can therefore be considered as function which receives an input u and returns an output $x = Cy$, where C

averages multiple user-selected node values from y to form an output value, at a requested discrete communication point in the future. Hence, we consider u as a vector of boundary values or source term value at specific nodes or elements of the mesh.

Figure 2 Lightweight FEM-FMU Scenario



Throughout this work, we use a time-dependent diffusion equation with Dirichlet boundary conditions:

$$\begin{aligned} \dot{T} - \varepsilon \Delta T &= f(u) \text{ in } \Omega \\ T &= \bar{T}(u) \text{ on } \partial\Omega \\ T &= T_0 \text{ in } \Omega \text{ for t} = 0, \end{aligned}$$

where T is the unknown solution, ε is a material parameter and f is the source term. A common application example for this type of equation is the heat equation. In this case T would be the temperature, $\bar{T}$ the temperature at the boundaries and T_0 the initial temperature. After the standard FEM space discretization processing as it is described in common textbooks concerning the FEM, see e. g. (Brenner and Scott, 2008) we end up with the sparse – and in general stiff – ODE or DAE system

$$\begin{aligned} M\dot{y} + Ay - b(u) &= 0 \\ \text{y} &= y_0 \text{ for t} = 0. \end{aligned} \qquad (1)$$

For the generation of this system we extended the open source software discussed in (Bernst, I. et al. 2016). The entries of the vector y now denote the approximation values of T at the nodes of the mesh that discretizes Ω. The dimension of this linear system can be very large, with which we mean hundreds of thousands. These resulting equations above are not limited to diffusion models. In the implementation used for the tests later on, linear elasticity is implemented as well and other models are possible.

Note that we assumed ε to be time-constant and f and the boundary conditions to be time-dependent from inputs. The boundary conditions are not limited to Dirichlet conditions, but can also be Neumann conditions. Here we restrict ourselves to Dirichlet boundary conditions for the sake of simplicity. Having a constant mesh allows to avoid a reassembling of the resulting linear system. With these restrictions only b changes according to the inputs u in a predefined way without requiring information about the grid at runtime:

$$b(u) = b_0 + Du$$

where b_0 is the base value (assembled with $u = 0$) and D the input factor matrix. In a model where Robin boundary conditions or ε should be time-dependent from input a similar approach could be done with $A(u)$, which is not considered here for the sake of simplicity.

So, writing a generator for this lightweight approach basically only requires to handle inputs and outputs and to export the system matrices M and A and vector b considering the inputs. The only real hard constrain of this lightweight approach is the constant mesh. A non-constant mesh would lead to the requirement to change and reassemble the FEM matrices during the simulation. In this case we end up with an extensive FEM-FMU scenario instead of lightweight one. Instead of including basically only the numeric solving it would become

necessary to include the whole logic of an FEM tool into the FMU. For example if one wishes to change the mesh during the simulation for the purpose of adaptive error control or for physical reasons this would require to include the initial mesh or geometry, the assembling routines, mesh refinement routines etc. In this case tool coupling via a CS FMU or the approach used by Stavåke, K., et al. (2013), is more reasonable.

3. CASE STUDY ON DIFFERENT TIME-INTEGRATION APPROACHES

After the basic assembling, the way of imposing the Dirichlet conditions determines whether the linear system (1) is a DAE or an ODE system, which reads

$$\begin{pmatrix} M_{II} & M_{ID} \\ 0 & 0 \end{pmatrix}\begin{pmatrix} \dot{y}_I \\ \dot{y}_D \end{pmatrix} + \begin{pmatrix} A_{II} & A_{ID} \\ 0 & I \end{pmatrix}\begin{pmatrix} y_I \\ y_D \end{pmatrix} = \begin{pmatrix} b_I \\ g \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} M_{II} & M_{ID} \\ 0 & I \end{pmatrix}\begin{pmatrix} \dot{y}_I \\ \dot{y}_D \end{pmatrix} + \begin{pmatrix} A_{II} & A_{ID} \\ 0 & I \end{pmatrix}\begin{pmatrix} y_I \\ y_D \end{pmatrix} = \begin{pmatrix} b_I \\ g + \dot{g} \end{pmatrix},$$

respectively, where the Index *I* means *inner states*, the Dirichlet constraints in the unknowns are denoted by y_D and in the known right hand side by g. The index *II* means *inner states tested at inner nodes*, *ID* means *Dirichlet node states tested at inner nodes*. Both kinds of systems tend to be stiff, so concerning time integration it is strongly advisable to use integrators that are able to handle stiff systems. Because of this and the matrix *M* any integrator will have to solve huge sparse linear systems in every iteration. Just like the back-end of the Modelica tools most standard integrators like CVODE or IDA from SUNDIALS – see (Hindmarsh A. C., et al. 2005), (Hindmarsh A. C. and Serban R., 2015) and (Hindmarsh A. C., et al. 2012) – do not have optimized linear solvers for these kind of systems. Nevertheless, the open source SUNDIALS software provides an API to connect suitable external linear solvers. Examples for such solvers are SuperLU which uses a BSD license or UMFPACK which uses now a GPL license and is well-known because of its usage in MATLAB.

Next, we take a look at the Backward Differentiation Formula (BDF) method as time integrator, which is implemented in CVODE and IDA. The BDF method is implicit and well suited for stiff systems as the ones resulting from space discretization of FEM models. Generally, BDF methods can be written for time step *n* as

$$\sum_{i=0}^{q} \alpha_{n,i} y_{n-i} = h_n \beta_n \dot{y}_n \;\Leftrightarrow\; \sum_{i=0}^{q} \alpha_{n,i} y_{n-i} - h_n \beta_n \dot{y}_n = 0,$$

where the order q is variable as well as the step size h_n. The coefficients α and β depend on q and h_n. CVODE and IDA vary both, but limit the order q to 5. We will now consider the different points of view of CVODE and IDA on ODEs or DAEs, respectively and their implications.

3.1 CVODE: Explicit ODEs

CVODE considers initial value problems (IVPs) as explicit ODE systems with initial values. So we choose the ODE formulation of the system. That means for the considered problem

$$\dot{y} = M^{-1}(b - Ay) \coloneqq f(t, y).$$

The FMU framework implements a C-function for f that connects to an external linear solver. CVODE will call this function and use the result to evaluate a function G, which is the negative left hand side of the BDF formula given above with $\alpha_{n,0} = -1$:

$$G(y_n) = y_n + h_n \beta_n \dot{y}_n - \sum_{i=1}^{q} \alpha_{n,i} y_{n-i} = y_n + h_n \beta_n \underbrace{M^{-1}(b - Ay)}_{=f(y,t)} - \sum_{i=1}^{q} \alpha_{n,i} y_{n-i}$$

For the built-in Newton method the iteration

$$J_G\left[y_n^{(m+1)} - y_n^{(m)}\right] = -G\left(y_n^{(m)}\right)$$

with the Jacobian

$$J_G = \frac{dG}{dy} = I + h_n \beta_n \frac{df}{dy} = I - \gamma_n M^{-1} A$$

and $\gamma_n = h_n\beta_n$ CVODE provides γ_n and $-G\left(y_n^{(m)}\right)$. The framework implements the C-function that solves this equation with a linear solver and also provides or approximates the Jacobian J_G. It is essential to handle the term $M^{-1}A$ appropriately. In the following subsections we will consider different ways how that can be done. Note that although M and A are sparse, $M^{-1}A$ and thus J_G are not sparse. Neither a full direct evaluation of J_G nor numerical differentiation to evaluate J_G would be feasible. This is a common problem and the solution is to avoid calculating an inverse matrix by using linear solvers if appropriate.

1. Solve a modified System

One way to deal with the term $M^{-1}A$ is to modify the equation such that the inverse vanishes:

$$\begin{aligned} & [I - \gamma_m M^{-1}A]\left[y_n^{(m+1)} - y_n^{(m)}\right] = -G\left(y_n^{(m)}\right) \\ \Leftrightarrow \quad & [M - \gamma_m A]\left[y_n^{(m+1)} - y_n^{(m)}\right] = -MG\left(y_n^{(m)}\right) \end{aligned}$$

This reformulation is possible within the CVODE interface, since the framework solves this equation. However, on the right hand side we see a term that analytically would vanish to identity:

$$MG(y_n) = My_n + \gamma_n M \underbrace{M^{-1}(b - Ay_n)}_{=f(t,y_n)} - M\sum_{i=1}^{q} \alpha_{n,i} y_{n-i}$$

However, since it is only possible to provide the evaluation of $\boldsymbol{f(t, y_n)}$ and not of $\boldsymbol{G\left(y_n^{(m)}\right)}$, the linear solving for $\boldsymbol{f(t, y_n)}$ cannot be avoided and induces some numerical errors as well.

2. Sparse approximate Jacobian

A more complex approach is to use a sparse approximate inverse algorithm (SPAI), see (Grote M. J. and Huckle T. 1996) to find an approximate inverse $\widetilde{M}^{-1}$ of the mass matrix *M*. In principle the accuracy can be controlled by minimizing $|| M\widetilde{M}^{-1} - I ||$ to a fixed tolerance. This could compensate different system sizes and higher fill-ins of *M*. Here we use an approach that preserves the sparsity pattern of *M* in its approximate inverse $\widetilde{M}^{-1}$. Therefore the achieved accuracy is limited by its fill-in structure. Then the approximation

$$J_G \approx I - \gamma_n \widetilde{M}^{-1} A$$

is used in the Newton-iteration.

3.2 IDA: Implicit ODEs / DAEs

IDA considers IVPs in an implicit form with initial values, i.e.

$$F(t, y, \dot{y}) = 0 \text{ with } y(t_0) = y_0 \text{ and } \dot{y}(t_0) = \dot{y}_0.$$

Thus we use the DAE formulation and fit our linear system into this form as follows:

$$F(t, y, \dot{y}) = M\dot{y} + Ay - b = 0.$$

The FMU framework implements a C-function, which computes this residual. For that IDA provides the current y and $\dot{y}$. In an implicit approach it is necessary to compute additional initial values $\dot{y}_0$ before the very first step can be computed. For the provided t_0 and y_0 these must satisfy

$$F(t_0, y_0, \dot{y}) = 0.$$

Therefore this consistent initialization of the system requires a little additional effort, which is however compared with the whole computational costs not significant.

Now, the relevant function for the Newton-method is simply F, where the BDF approximation replaces $\dot{y}$:

$$G(y_n) \coloneqq F\left(t_n, y_n, \frac{1}{h_n \beta_n} \sum_{i=0}^{q} \alpha_{n,i} y_{n-i}\right) = 0$$

Thus the iteration is

$$J_G\left[y_n^{(m+1)} - y_n^{(m)}\right] = -G\left(y_n^{(m)}\right)$$

with the Jacobian

$$J_G = \frac{dG}{dy} = \frac{\partial F}{\partial y} + \tilde{\gamma}\frac{\partial F}{\partial \dot{y}} = A + \tilde{\gamma} M,$$

where $\tilde{\gamma} \coloneqq \frac{\alpha_{n,0}}{h_n \beta_n}$. Note, in this context J_G is sparse and easy to calculate. The framework implements the C-function, which connects to an external linear solver for the iteration and also calculates the Jacobian J_G. It receives among other things $-G(y_n)$ and $\tilde{\gamma}$ as arguments from IDA.

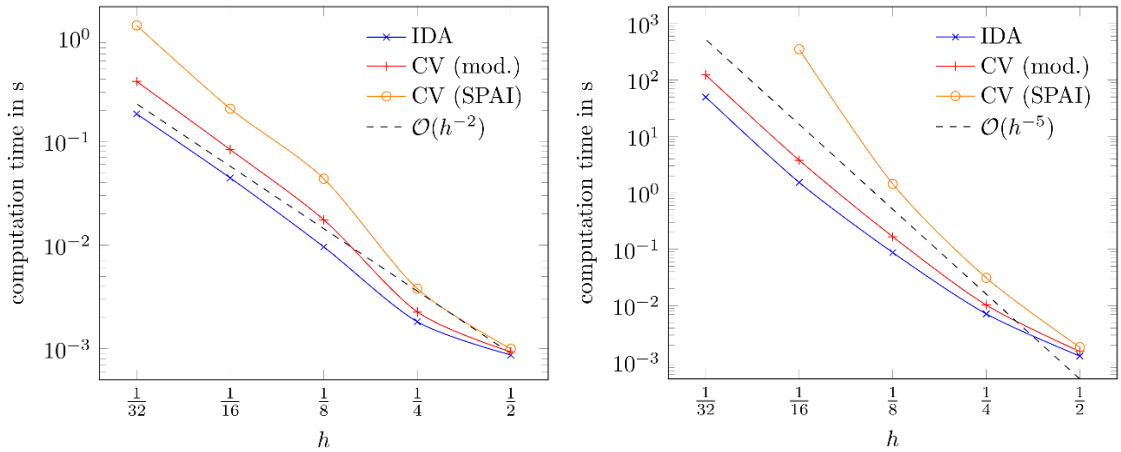
4. RESULTS

In this section the performance of the different methods is compared. For this we use FEM with linear triangular elements in 2D and linear tetrahedral elements in 3D. In 3D the system has way more fill-in due to more connections. For any linear system solving UMFPACK is used, since it is well suited and fast for the sparse and non-symmetric matrices we encounter in our FEM implementation.

A simulation master algorithm belongs to the simulation tool, where the FMU is simulated in. So it can be different in every tool. Therefore we setup a single FMU test benchmark to negotiate effects from different master algorithms. This allows to measure the performance of the FMU itself. Also, the internal integrator of the FMU can therefore choose the time steps adaptively without considering the synchronization with other units. In the tests all time-integrators have been used with the same tolerance settings.

First, we choose a 2D problem with a smooth, differential solution on the unit square. The unit square was meshed regularly with different minimal geometric step sizes h. Next, we use an analogous problem in 3D with the unit cube. For these tests the simulation end time is set to 20. The computation times are measured for the described methods and shown in the Figure below. If one halves h in 2D the number of unknowns N increases quadratically, in 3D cubically. The size affects the linear solving: As discussed e. g. in (Collier N. et al. 2012) for direct solvers executed on regular FEM meshes in 2D one can expect a growth of the complexity of $\mathcal{O}(N^{1.5})$ and for three dimensional problems of $\mathcal{O}(N^2)$. If we transfer this in a notation depending on h we have $\mathcal{O}(h^{-3})$ and $\mathcal{O}(h^{-6})$, respectively. UMFPACK is using the unsymmetric multifrontal method.

Figure 3 Performance of different approaches in 2D (left) and 3D (right) for different mesh sizes

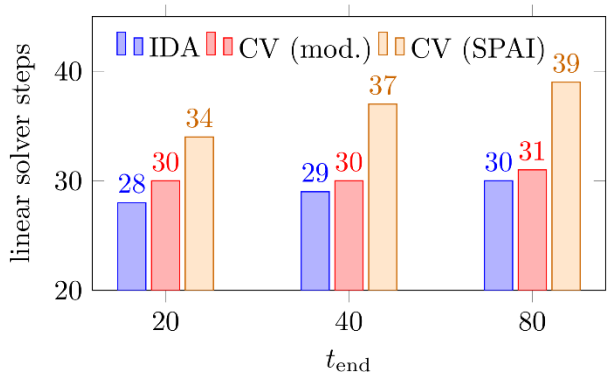


The observed behavior in Figure 3 is better than one may expect, but under good conditions UMFPACK can perform better than common direct solvers, see e. g. (Liu, Haixin, and Dan Jiao 2010). For the test problems method *IDA* performs best. In 2D it almost reaches a behavior of $\mathcal{O}(h^{-2})$ in performance, in 3D almost $\mathcal{O}(h^{-5})$.

The method *CV (mod.)*, which stands for *CVODE* with modified system, is the next best method and shows for 2D and 3D the same asymptotic behavior as *IDA*. However, the computation times of *CV (mod.)* are larger by a constant factor, which makes this methods less efficient. This factor might arise from the additional linear solving in each step to evaluate $f(t, y_n)$, which is used in the linear system of the Newton iteration.

The *CV (SPAI)* shows only for 2D the same asymptotic behavior as IDA. There, the performance factor is even worse than for *CV (mod.)*. This might arise from the building of the approximate inverse. Yet, the Jacobian seems to be accurate enough for the Newton iteration to converge immediately. This is not the case for the 3D problem. *CV (SPAI)* cannot keep up the same order as *IDA* and *CV (mod.)* for larger grids. One reason might be a bad accuracy for the approximate inverse $\widetilde{M}^{-1}$ due to the preservation of the fill-in structure of M. With this setting it seems not be possible to handle larger systems in combination with a higher fill-in due to more connectivity in a 3D mesh than in a 2D mesh. Also models with vector solutions like linear elasticity have again a higher fill-in due to the dependencies across the spatial dimensions and might suffer from the fixed fill-in structure. A possible solution would be to use an implementation with a more flexible approach for the fill-in of $\widetilde{M}^{-1}$ as discussed e. g. in (Grote M. J. and Huckle T. 1996).

Figure 4 Total number of linear solves of different schemes in 2D simulations with different simulation end times and fixed geometric step size.



Now we perform a different test with a fixed geometric step size to $h = \frac{1}{8}$ and consider the same simulation but vary the simulation end times, namely 20 s, 40 s and 80 s. Figure 4 shows the result. One can see that although the time doubles, the number of linear solver steps approximately increases linearly for *IDA*, *CV (mod.)* and *CV (SPAI)*. This indicates an exponential increase of the time step width, which is expected for this simple, converging problem.

5. CONCLUSION AND FUTURE PROSPECTS

To use the full capabilities of a common model format like the Functional Mock-Up Unit for models based on partial differential equations it is very important to make sure that such a unit can really be distributed as stand-alone package. Approaches that depend on a FEM tool being available for all users of the FMU subvert the idea of interoperability and intellectual property protection for modeling and simulation know-how.

We proposed a lightweight FMU framework based on open source software. This allows to create FMUs for FEM models that can be run independent of additional tools and thus also independent of license restrictions. Such an FMU can be easily distributed to cooperating companies. Additionally, due to its lightweight design, it is very easy for an existing tool to generate the missing model dependent part to complete the framework to gain the FMU.

In a case study we discussed several options to implement FEM models within the FMI concerning implementation and efficiency from a numerical point of view. The discussed options include methods with an explicit view and one method with an implicit view (IDA) on the linear system. From the tested approaches

the one based on *IDA* software from SUNDIALS showed very promising results. For further work, a more specialized real-time capable custom implicit Euler time-integrator will be tested and compared to IDA, especially for 2D use cases with constant matrices or 1D use cases. Exploiting the system structure in the time-integrator could avoid some operations, since a Newton iteration is actually not required. Also the consistent initialization procedure is only required because of the implicit view on the DAE system.

REFERENCES

Arnold, M. et al, 2014. Error analysis and error estimates for co-simulation in FMI for Model Exchange and Co-Simulation v2.0. In S. Schöps, A. Bartel, M. Günther, E.J.W. ter Maten, P.C. Müller (eds.): *Progress in Differential-Algebraic Equations*. Springer Berlin Heidelberg, pp. 107-125.

Åkesson J. et al, 2012. Generation of sparse Jacobians for the Function Mock-up Interface 2.0. *Proceedings of the 9th International Modelica Conference*. Munich, Germany, pp. 185-196.

Blochwitz, T. et al, 2011, The functional mockup interface for tool independent exchange of simulation models. *Proceedings of the 8th International Modelica Conference*. Dresden, Germany, pp. 105-114.

Brenner, S. and Scott L., 2008. *The mathematical theory of finite element method, 3rd ed.* Springer-Verlag, New York-Berlin-Heidelberg.

Bernst, I., Kaufmann, C. and Frochte, J, 2016. On Learning Assistance Systems for Numerical Simulation. In *IADIS - International Journal On Computer Science And Information Systems*, Vol. 11, No. 1, 115-133.

Cellier, F. E. and Kofman, E., 2006. *Continuous system simulation*. Springer Science & Business Media.

Collier, N. et al, 2012. The cost of continuity: A study of the performance of isogeometric finite elements using direct solvers. In *Computer Methods in Applied Mechanics and Engineering*, Vol. 213-216, pp. 353-361.

Davis, T. A., 2004. Algorithm 832: UMFPACK V4.3---an unsymmetric-pattern multifrontal method, In *ACM Transactions on Mathematical Software (TOMS)*, Vol. 30, No. 2, pp 196 - 199.

Fritzson, P., 2004. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press.

Grote, M. J. and Huckle, T., 1997. Parallel preconditioning with sparse approximate inverses. In *SIAM Journal on Scientific Computing (SISC)*, Vol. 18, pp. 838-853.

Hindmarsh, A. C. et al, 2005. Sundials: Suite of nonlinear and differential/algebraic equation solvers. In *ACM Transactions on Mathematical Software (TOMS)*, Vol. 31, No. 3, pp. 363-396.

Hindmarsh, A. C. and Serban, R., 2015. *User documentation for cvode v2.8.2*, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory.

Hindmarsh, A. C. et al, 2012. *User documentation for IDA v2.7.0*, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory.

Khan, S., 2012. *Discretizing PDEs for MapleSim. adept scientific* plc.

Liu, H. and Jiao, D. 2010. An H-LU Based Direct Finite Element Solver Accelerated by Nested Dissection for Large-scale Modeling of ICs and Packages. *PIERS Proceedings*, Cambridge, USA, pp. 774-778.

Modelica Association, 2014, *Functional Mock-up Interface for Model Exchange and Co-Simulation 2.0*, c/o PELAB, IDA, Linköpings Universitet, Linköping, Sweden

Stavåker, K. et al, 2013. PDE Modeling with Modelica via FMI import of HiFlow3 C++ Components. *Proceedings of SIMS 54th Conference SIMS Conference on Simulation and Modeling*. Bergen, Norway.

Stavåke, K. et al 2014. PDE Modeling with Modelica via FMI import of HiFlow3 C++ Components with Parallel Multi-Core Simulations. *Proceedings of the 55th SIMS Conference on Simulation and Modeling*. Aalborg, Denmark.

Li, X. S., 2005. An Overview of SuperLU: Algorithms, Implementation and User Interface. In *ACM Transactions on Mathematical Software (TOMS)*, Vol. 31, No. 3, pp. 302-325.

Li, Z. et al, 2008. Solving PDE models in Modelica. In *2008 International Symposium on Information Science and Engineering*. Shanghai, China, pp. 53-57.