FMI Go! A simulation runtime environment with a client server architecture over multiple protocols

Claude Lacoursière, claude@hpc2n.umu.se

Tomas Härdin, tomas.hardin@umu.se
HPC2N/UMIT, Umeå University
SE-901 87, Umeå, Sweden

Abstract

We present a distributed software infrastructure to perform distributed simulations with Functional Mockup Interface (FMI) compatible components. The current implementation supports both TCP/IP and MPI. This is a client-server design where the client is the global simulation stepper and the servers are the simulation modules. Features on the master time stepping algorithm currently include several time stepping algorithms including one which can handle algebraic constraints, root finding for cases involving loops, and support for asynchronous data exchange with "monitors" and "observers" which only consume data. The servers provide support for numerical directional derivatives, filtering, and interpolation. Support is provided for the System Specification and Parameterization (SSP), an emerging standard aimed at supporting the FMI.

The software is open source with a permissive license and designed to be used inside simulation environments and platforms with user interfaces. The focus being on the mathematical and runtime aspect of FMI based simulations.

1 Introduction

No one simulation tool can satisfy everyone's needs and yet, full system simulation is the order of the day. Models created using different tools must be made compatible with each other for data transfer at least, and by force of reality, a lowest common denominator must be found for numerical time integration of modular, heterogeneous systems. In this model, subsystems are black boxes connected with simple elements representing boundary conditions. The (FMI)(MODELISAR, 2014) standard specifies an API which answers the first question of data formats as well as fundamental functionality to initialize and terminate modules, and defines semantics to handle events etc. However, this standard does not specify the requirements on the runtime environment or the master stepper.

We consider both these issues with the aim of providing a minimal runtime infrastructure which is fully standards compliant as well as open. We also intend to develop a number of numerical methods for time integration. This should allow academics to test their new numerical methods on nontrivial examples. The hub based design should also allow people to write their own interfaces to connect with the data analysis and visualization tools they prefer, and can serve as a foundation for commercial integration tools with sophisticated user interfaces.

In what follows we describe the nature of the problem we are trying to resolve in Sec. 2, then cover some previous work in Sec. 3. We then describe some details of our architecture in Sec. 4. Force based model coupling and is described in Sec. 5 and a kinematic coupling as well as a differential algebraic stepper is found in Sec. 6. Experiments and discussions are in Sec. 7,8,9 and in Sec. 10.

2 Problem statement and objectives

Software tool interoperability requires a standardized interface to be implemented by vendors, as well as a standard format to describe and exchange source or binary code implementing a Functional Mockup Unit (FMU). Also needed mathematical model which corresponds to the interface, a configuration format and editor for producing configuration files. Then comes a runtime environment which can read these, load the FMUs and perform time integration. One also needs data collection from the runtime environment, data formats and communication protocols. Of course, one also needs one needs numerical methods for time integration. When all this is in place, one can create simulations, run them, gather data, and analyze it with the tools of their choice.

The FMI specifies only the first three items: interface, exchange formats, and high level mathematical formulation. The emerging standard System Specification and Parametrization (SSP) (Köler et al., 2016) aims at defining the structure of a simulation – which FMU connects to which and on what port – as well as parameterization, including unit conversion etc. This is in the process to be adopted by the FMI committee. Editors for SSP are under development by vendors. There is also a Software Development toolKit (SDK)(QTronic, 2017) which is a reference implementation of the FMI API and can serve as a foundation for writing runtime environments.

We decided to develop components of the runtime environment including SSP, protocols and formats for data communication and handling, as well as stepping methods. We believe that these are the components missing to

achieve a genuinely modular solution which avoids vendor lock-in, as well as a sufficiently complete environment for researchers to test their numerical methods and simulation master algorithms. We are also considering IP and secrecy issues which can be supported with our client server design. Of course, we believe that performance is a fundamental aspect.

Loading many shared libraries as is suggested by the FMI documentation always leads to problems. This is why we chose a client server architecture which we implemented over TCP/IP for LAN and WAN configurations, and MPI for standalone of cluster ones.

User interfaces we leave for others.

The objective is to provide a robust runtime environment with good numerical methods which goes from SSP to data files, based on standards and protocols so that visualization and analysis software of choice can be plugged in easily.

3 Previous work

Nearly twenty FMI *import* tools are listed on the FMI website (MODELISAR, 2014) and these fall unevenly into two categories. First come the well established simulation packages which support import functionality in order to connect to third party tools. Second come *integration* tools only designed to couple simulation and analysis tools, which is close to our own work. These divide further into commercial and open source ones. Of the latter, DACCOSIM (Galtier et al., 2015) is the closest to our effort.

Yet all integration tools we know of aim at providing a full environment and it appears that the issue of the quality of time integration is secondary at best, yet the numerical methods are locked down which isn't good if one has a particularly recalcitrant and difficult model. We don't see the need to keep numerical methods secrets. And this prevents experimentation on real-life problems by academics. One of our motivations.

Distributed and modular simulations isn't new and predecessors include the High Level Architecture (HLA) (IEEE, 2010) for instance, which has even been adapted to FMI (Awais et al., 2013). Focusing on FMI compatible efforts, Ptolemy (Ptolemaeus, 2014) is an object oriented peer-to-peer agent based simulation environment and has now FMI (Broman et al., 2013; Cremona et al., 2016) capabilities with an eye on meeting the requirements for discrete-continuous simulations (Zeigler, Praehofer, and Kim, 2000), and DACCOSIM (Galtier et al., 2015) which is most similar to ours. There are others yet but too numerous to list here.

Why a new effort? First because there is a need for a test environment for new time integration methods which is not possible with commercial tools. The open source projects did not seem to have this as a focus.

Then comes a more contentious issue: software license. The commercial dimension here weights heavily so we chose the permissive MIT (MIT, nodate) license to avoid any problem.

It is clear from the literature about time integration for cosimulations methods (Fiedler and Arnold, 2014; Martin, Christoph, and Tom, 2013; Schierz, Arnold, and Clauss, 2012; Schierz and Arnold, 2012; Arnold, 2010; Bernhard Schweizer, Li, and Daixing Lu, 2015; Bernhard Schweizer, Daixing Lu, and Li, 2015; B. Schweizer and D. Lu, 2015) that it is hardly possible to control errors or reach stability without having access to directional derivatives or at least the ability to rollback which isn't available in too many cases. However, both of these features can be provided by the runtime environment with numerical differentiation and various brute force techniques. This is clearly not addressed in any of the tools we looked at. One can provide these features when wrapping a ME FMU into a CS FMU, since one, which means that it might be advantageous to export as ME when possible and let a wrapper take care of more advanced features.

The DAE stepper presented in Sec. 6 is different from that of Schweizer (Bernhard Schweizer, Daixing Lu, and Li, 2015) in that we are using a previously published relaxation and regularization technique (Lacoursière, 2007) which is provably linearly stable, unlike the variants Schweizer analyzed (Ascher and Petzold, 1993). Experience has proved that our method does not require the solution of nonlinear systems of equations as the linearized approximation is sufficiently stable and produces no systematic drift.

Therefore, we believe that our work has much orthogonality with what already exists, enough to add yet one more FMI runtime environment to the list.

4 Software design

We opted for a client-server architecture in which each server process hosts an individual FMU. The global stepper is then a client, consuming results produced by the FMUs, and serves also as a data hub. It is also a server to *monitors* which are read-only applications for interactive, online visualization and data analysis, as well as data storage. See Fig. 2. We decided against peer-to-peer communication

We used Protobuf (*Protocol Buffers* 2017) to map the twenty or so functions in the FMI API to messages which can be passed via ZeroMQ (iMatrix, 2017). The servers dynamically load an FMU and using the QTronix SDK (QTronic, 2017). The same was repeated to use the Message Passing Interface (MPI) (MPI, 2017) which has the benefit of not needing to pack and unpack data. We chose to use (MPICH, 2017) because it is better than other implementations at handling oversubscription, i.e., when there are more processes than cores available. On Windows, we use the native library (Microsoft, 2017).

The numerical Jacobians were implemented in the servers using simple first order finite differences. These computations are done in the servers which can exploit

DOI 10.3384/ecp17132653

parallelism to perform this task. This requires the ability to rollback a simulation one or several times, and if possible, to clone it for parallelism. In the best case, FMUs provide rollback functionality. The second best case is a functional serialization and deserialization. We are investigating other methods, as well as the trade off between doing much more work per step vs step size and accuracy.

Support for Model Exchange (ME) FMUs is in development on two fronts. One is a global stepper capable of handling ME FMUs or combinations of ME and CS FMUs, i.e., integrating discrete and continuous systems(Zeigler, Praehofer, and Kim, 2000). The other is to include a local ME stepper inside the servers so that ME FMUs can be transformed to be CS ones. The advantage here is that even though CS export tools make a choice of numerical time integration which cannot be changed, yet is critical for stability and application dependent. Therefore, delaying the decision until runtime is appealing. In addition, ME FMUs are required to be able to cancel a step as long as no event is crossed which makes it easier to compute numerical derivatives and rollback. Access to the ME FMU also allows the support of extrapolation and interpolation methods (Bernhard Schweizer, Li, and Daixing Lu, 2015) after the model has been exported, and to introduce other types of filters on the inputs (M. Benedikt et al., 2013; Drenth, 2016). We have already automated the latter. Briefly, assuming a module has continuous states x, inputs u and outputs $y = g(x,u)$, the dynamics is augmented so that

$$\begin{aligned} \dot{x} &= f(x,u,t) \\ \dot{z} &= x, \end{aligned} \tag{1}$$

and then the reported output is

$$y = \langle g(\langle\langle x \rangle\rangle, \langle\langle u \rangle\rangle) \rangle \tag{2}$$

instead of $g(x,u)$ at the end of the communication step. The averages can be, e.g.,

$$y = g\left(\frac{1}{2H} (z_0 + z_1), \langle\langle u \rangle\rangle) \rangle \right), \tag{3}$$

where H is the communication step, and z_0, z_1 are the values of z at the beginning and end of the communication step (Drenth, 2016). The advantage of such filters is to offset the noise produced by the discontinuous inputs at each communication point. Such functionality is clearly not possible when using a CS FMU, and we believe that it is good to leave the choice open. We have automated the augmentation of the equations of motion. Other types of filters are straight forward to implement.

The overall design is shown in Fig. 1. Here, one goes from a FMU via the Qtronix library to the FMI API. At this point, depending on whether the FMU is ME or CS, support libraries are used to deliver additional functionality to the global stepper. This funnels through the FMI/X communication library towards the global stepper, and said returns results to be processed by the FMU.

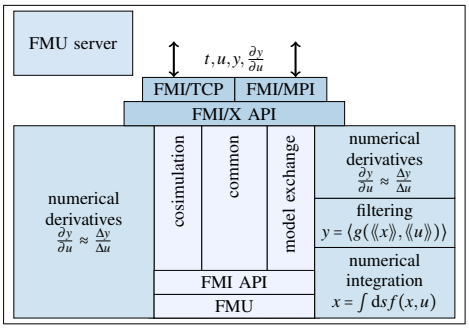

Figure 1. Server architecture

The global stepper program has a barebone, command line interface to describe the connections, as well as support for SSP files which is much more convenient.

The global stepper can also resolve loops at initialization using Newton-Raphson's method, and we intend to use this feature for the ME stepper so that it can process DAEs.

As the kinematic stepper requires the solution of linear problems, we currently use UMFPACK (Davis, 2004).

The overall design of the system appears diagrammatically in Fig. 2.

For TCP/IP, there are a number of issues related to the GRID computing concept of "network weather service", which is about resource discovery and allocation. This is not implemented yet but there are simple tools for this.

Hardware in the Loop (HIL) functionality has not been developed at this time though the architecture is compatible with this.

The software runs on Linux, Mac OS X and Windows.

To emphasize, this type of design is not entirely novel as mentioned in Sec. 3. What is different however is the restriction we imposed ourselves to the runtime environment and not the user environment. In addition, the existing functionality and what is in planning will hopefully provide building blocks for the development of new numerical time integrators, since typical restrictions of FMUs – absence of directional derivatives or rollback functionality – will be compensated for by the support modules, as described above, i.e., wrappers for ME FMUs to transform them into CS FMUs.

We follow the UNIX philosophy here: "Do one thing and do it well". For our case, the pipe model is "initial conditions in, data out". Clearly, there is more than one thing going on here, but we are aiming at being atomic and modular: all that's needed to perform time integration of systems made of FMUs, but only that.

To confirm that this is a position statement, we believe that numerical algorithms have little if anything to do with trade secrets, yet should be tested extensively in real situations. When an engineer runs a simulation, the same wants to have confidence that the results make sense. For that reason, the numerical time integration software should open. If successful, the best methods will be available for all to use.

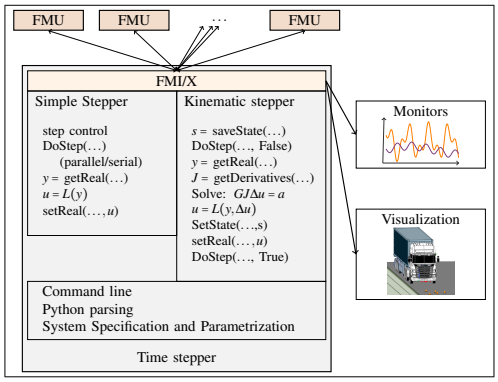

Figure 2. Overall architecture

5 Force subsystem couplings

Splitting a system leads to having one variable, x, say, which appear in two subsystems. For instance, the output shaft of an engine is the very same as the input shaft of a clutch so the angle of said should be the same in each module.

As the systems are integrated independently, the duplicates, $x^{(1)}, x^{(2)}$ cannot remain in sync. One strategy for physical systems at least is to introduce a generally stiff spring-damper in one or both subsystems. There are several choices as described previously (Bernhard Schweizer, Li, and Daixing Lu, 2015) namely, force-displacement or force-velocity in which one of the units contains a spring-damper but not the other. These we call "holonomic" and "non-holonomic", respectively. Then there is displacement-displacement coupling in which case there are spring and dampers on both sides. Finally, there is the "spring free" case described in Sec. 6 which requires a global solver to computes the force required so that $x^{(1)} = x^{(2)}$ at each communication step, an algebraic condition. The latter requires both rollback and directional derivatives which is not often supported. Rollback is also required for "iterative" methods which are essentially fixed point iterations (Bernhard Schweizer, Li, and Daixing Lu, 2015) and have good stability properties. As mentioned, we aim at making our software capable of applying these methods for any FMU, whether it has these features natively or not.

We chose force-velocity in our examples.

Whenever there is a spring-damper at the input of a system, system 1, say, there is an input signal

$$u^{(1)} = x^{(2)}, \text{ or } u^{(1)} = v^{(2)} \text{ or both (with abuse of notation).} \tag{4}$$

But this signal is not available continuously, only at the beginning once per communication step. This values are denoted as $\bar{x}^{(2)}$ and $\bar{v}^{(2)}$. The coupling force, given spring and damping constants $k^{(c)}$, $\gamma^{(c)}$, respectively, is then

$$f^{(c)} = -k^{(c)}(x^{(1)} - \bar{x}^{(2)}) - \gamma^{(c)}(x^{(1)} - \bar{v}^{(2)}). \tag{5}$$

The reaction force $-\bar{f}^{(c)}$ at the end of the reported to the coupled element. When $\bar{x}^{(2)}$ is not reported, the approximation

$$x(t) - \bar{x} \approx -\int_0 \mathrm{d}s(v - \bar{v}) \tag{6}$$

is used and $x(t) - \bar{x}$ is reset to 0 at the beginning of each communication step.

A variety of methods can be used to improve on the Zero Order Hold (ZOH) including extrapolation, or combination of extrapolation and interpolations, often called iterative methods (Bernhard Schweizer, Li, and Daixing Lu, 2015, and references therein).

One thing remains though, the spring-dampers $k^{(c)}, \gamma^{(c)}$ are not in the original model and introduce artificial dynamics. One needs to keep the frequencies due to the couplings much higher than the *design* frequencies, i.e., time scales involved by the couplings should me much smaller than those of interest in order to not interfere with the results as we show in Sec. 9, and this leads to communication steps which are orders of magnitude smaller than the time scales of interest, and introduces stiffness in the individual modules as they fight against the spring force in Eqn. (5). As we show below in Sec. 9, coupling frequencies need to be at least one order of magnitude above the design frequencies, meaning that coupling springs must be two order of magnitude above the stiffness of the internal force derivatives.

There are alternatives to this which do not involve a global solver as the one in Sec. 6, such as bilateral delay lines (TLM) (Dag Fritzson, 2007; Krus, 1995). This is still a spring-damper coupling but motivated by the fact that a force takes finite time to traverse any form of physical coupling, interactions are interpolated between the two previous steps. The main issue here is that this only works with intermediate steps within the `DoStep` calls. Something which can only be addressed with ME FMI using state machines without continuous states, one of our next steps. An effort similar to ours but based on TLM is in the process of being released to the public (Sjölund et al., 2010).

As far as trying to damp the high frequencies due to coupling and avoid oversampling the system, an antialiasling technique as been presented recently (Drenth, 2016) which is promising. We have introduced it into our software though we are not including this in our results as explained below in Sec. 11.

6 A differential algebraic stepper

As mentioned in Sec. 5, a split model involves algebraic conditions, and these can be taken care of directly by a DAE method (Bernhard Schweizer, Daixing Lu, and Li, 2015; Bernhard Schweizer, Li, Daixing Lu, and Meyer, 2015; Bernhard Schweizer and Li, 2015; B. Schweizer and D. Lu, 2015), though that requires rollback and directional derivatives. There are no spring-dampers in this model and therefore, no parasitic dynamics. Also, the problems related to choosing suitable spring and damping constants for the couplings is now entirely avoided, so are

DOI 10.3384/ecp17132653

artifacts introduced by the numerical integration methods because of stiffness, or unnaturally small time steps.

We reuse ideas from multibody dynamics (Lacoursière, 2007) to design a stepper which computes the interaction forces required to maintain the constraints at least linearly, and uses damping to stabilize the symmetric average of the algebraic conditions.

Consider two systems with output variables $y^{(1)}, y^{(2)}$ as well as time derivatives $\dot{y}^{(1)}, \dot{y}^{(2)}$. These variables are constrained either holonomically or nonholonomically, respectively, meaning that in discrete time we should have

$$g(y_k^{(1)}, y_k^{(2)}) = 0 \text{ or } q(y_k^{(1)}, y_k^{(2)}) = G^{(1)}\dot{y}_k^{(1)} + G^{(2)}\dot{y}_k^{(2)} = 0, \tag{7}$$

respectively, where k is the discrete time index. We also write $G = \partial g/\partial y$ so that $G\dot{y} = 0$ holds for both cases, abusing notation. First we make the assumption that

$$y_{k+1}^{(j)} \approx y_k^{(j)} + h\dot{y}_{k+1}^{(j)}, \tag{8}$$

where H is the communication step is a reasonable approximation. This is the case for the SHAKE (Hairer, Lubich, and Wanner, 2001) stepper and a variant of ours (Lacoursière, 2007). Note that $k+1$ is used on the time derivatives. Next we write a time translation operators as

$$y_{k+1}^{(j)} = \Phi_k^{(j)}(u_k^{(j)}) \text{ and } \dot{y}_{k+1}^{(j)} = \Psi_k^{(j)}(u_k^{(j)}). \tag{9}$$

The aim now is to compute $u_k^{(j)}$ in such a way that Eqn. (7) is satisfied at $k+1$. Assuming that all modules can rollback, we start with a guess $\bar{u}_k^{(j)}$ and from this we expand the constraint equations in Eqn. (7) to compute $u_k^{(j)} = \bar{u}_k^{(j)} + \delta u_k^{(j)}$ such that the constraints are satisfied. This requires the directional derivatives

$$\frac{\partial \dot{y}_k^{(j)}}{\partial u_k^{(j)}} = \frac{\partial \Psi_k^{(j)}}{\partial u_k^{(j)}}, \tag{10}$$

which are mobilities in the case of multibody dynamics, or admittance for electrical circuits. Dropping superscripts on all variables and writing G for the agglomerated Jacobian of the constraint equations and g_k for the value of the constraint equation at discrete time k, δu should satisfy

$$\begin{aligned} g_{k+1} &\approx g_k + hG\dot{y}_{k+1} \\ &= g_k + hG\Psi_k(\bar{u}_k + \delta u) \\ &\approx g_k + hG\dot{\bar{y}}_k + h\left[G\frac{\partial \Psi_k}{\partial u_k}\right]\delta u. \end{aligned} \tag{11}$$

This linear approximation can be stabilized as shown in our previous work (Lacoursière, 2007), which is unconditionally stable, unlike more popular methods (Ascher and Petzold, 1993) variants of which have been studied also in the context of cosimulation (Bernhard Schweizer, Daixing Lu, and Li, 2015). However, methods mentioned above all based on spring-damper ideas and introduce second order dynamics on the algebraic condition. Our method is of first order only and this is what provides stability. We also use a symmetric form of the constraint so that in fact, we are enforcing

$$\frac{1}{4}(g_{k+1} + 2g_k + g_{k-1}) + \frac{\tau}{h}G_k v_{k+1} + \frac{\varepsilon}{h}u_{k+1} = 0, \tag{12}$$

where τ is a relaxation time and serves as stabilization. This is clearly of first order in and of itself, but of course, the averaging does introduce oscillations, damped by the τ term. Such symmetric projections have been shown to have good energy preservation properties (Hairer, 2000) when $\tau = 0$ and the nonlinear equations are solved exactly. The parameter ε has the same unit of the inverse of a spring constant if we assume that u has units of force, and is there only to prevent against constraint degeneracy but can be shown to introduce physical compliance when τ is sufficiently small. However, τ serves as a low pass filter and when $\tau = 2h$, the oscillations are just below critical damping. This is needed to stabilize on the constraint manifold. This analysis is found in part in our own work cited above. With this parameterization, τ/h is the rate of exponential decay of the constraint violation. There is no such guarantee of stability with the standard scheme (Ascher and Petzold, 1993). A more thorough stability analysis is in preparation. We linearize Eqn. (12) to avoid having to solve the nonlinear system of equations. Introducing the parameter

$$\gamma = \frac{1}{1 + 4\tau/h} \tag{13}$$

we need to solve the following linear system of equations for δu

$$\left[G\frac{\partial \Psi}{\partial u} + \frac{4\gamma\varepsilon}{h}\right]\delta u = -\frac{4\gamma}{h}g_k + \gamma G\dot{y}^{(k)} - G\dot{\bar{y}}^{(k+1)} \tag{14}$$

In practice, one performs a step with some guess for inputs $\bar{u}_k$ to obtain a preliminary estimate on the velocities $\dot{\bar{y}}_{k+1}$, rollback, compute δu, and then step forward again with $u_k = \bar{u}_k + \delta u$. This has been shown to work very well even for nonsmooth, event driven systems (Lacoursière and Sjöström, 2014) dozens of units simulated in parallel.

Note here that if there are n observables $y \in \mathbb{R}^n$ and m control inputs $u \in \mathbb{R}^m$, the system is underactuated if $m < n$, overactuated if $m > n$ and fully actuated when $m = n$. The matrix $G\partial\Psi/\partial u$ has full row rank when $m \leq n$ but is degenerate otherwise, and this is where $\varepsilon \geq 0$ comes in to regularize the system.

The control flow is described in the following. In this notation, each statement is understood to be applied to all FMUs in parallel and all superscripts are removed.

s = `GetState(...)`
`SetXXX(...)`, $\bar{u}$
`DoStep(...,`$t, t+h$`, False)`
`GetReal(...,`$\bar{y}$`)`
`GetDirectionalDerivative(...,`$\partial\Psi/\partial u$`)`

Assemble system matrix $G\partial\Psi/\partial u$
Solve for inputs δu from Eqn. (14)
`SetState(...,`s`)`
`SetRealXXX(...,`$\bar{u}+\delta u$`)`
`DoStep(...,`t`, h, True)`

7 Experimental methodology

In evaluating the performance of FMIGo! we focused on the latency introduced by the FMI/X communication layer. We looked at scalability here and worked strictly with the best case scenario using loopback ports. Overhead due to TCP/IP routers or switches varies greatly, but are in the millisecond range.

For the rest, we compare both the accuracy of cosimulated systems with respect to reference or analytic solutions, and pay attention to the small time scales introduced by the spring-damper couplings.

Two examples are considered, chains of spring damper systems with uniform masses, and a simple truck model often used for elementary analysis (Eriksen and Nielsen, 2014). The latter example contains large mass ratios.

In all cases, we choose our time steps using dimensional analysis and compare them the periods of oscillations in the systems, the rationale being that accurate solutions should require around twenty steps per period of oscillation – the smallest in the system – , as is the case for most good numerical time integration methods as easily verified. For instance, an embedded Runge Kutta method of order 4/5 requires 15 steps per period for the simple harmonic oscillator to reach local tolerance of 10^{-4}, involving 90 function evaluations, though forward Euler requires at least 50 function evaluations (steps) to produce a reasonable solution, though more than 200 to deliver any form of accuracy. And because ZOH techniques are very akin to forward Euler, this is the best one can expect.

8 Timing

Measurements on 4 cores i7 2.8GHz, sufficient memory and a vanilla Linux installation. We had 4GB available, but the footprint of the program was small enough as to be irrelevant for common hardware. Results will vary for different systems but this should give a good idea of the overhead involved.

Using `/usr/bin/time` utility we extracted `user`, `system` and `wall` time. The `user` time is spent in computation and signal routing and communication packaging and a part of the time spent in moving data via sockets.

For TCP/IP version, `system` includes time for polling, waiting and going through the TCP/IP stack. For the MPI version, `system` includes interprocess communication which depending on the MPI library, might also go via sockets and TCP/IP. So, this time has to be considered in the present case as it is used by the application. There are other unrelated processes counted in `system` but that was made negligible by stopping all irrelevant applications.

The `wall` time is larger than the sum of `user` and

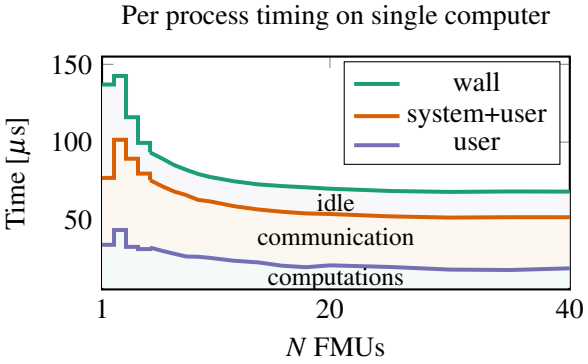

Figure 3. Timing measurements in loopback configuration

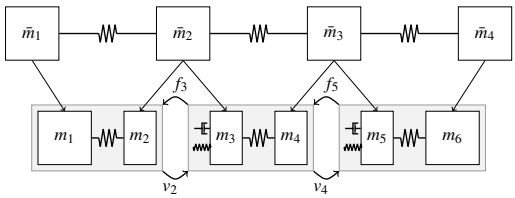

Figure 4. A chain of mass spring-dampers split into subunits

`system` because the CPU is not fully utilized and spends cycles waiting for packets and communication.

Note that this timing is sensitive to the choice of MPI library for the case of oversubscribing, i.e., when there are more processes than cores. OpenMPI performs badly in this case and seems to have quadratic complexity. MPICH however is well behaved and delivers linear performance as shown in Fig. 3. The Microsoft MPI library did perform well also.

Our experiment consisted of minimal FMUs which contained a point mass and a spring-damper. Computations were minimal and represent a lower bound on any practical simulation. What is therefore included here is all the time needed to perform time integration on said physical model, communication to the master stepper, routing of signals and communication back to the individual FMUs.

The conclusion is that a distributed design, at least when using MPI, is negligibly slower than one based purely on dynamic loading. The benefits of MPI however are immense as one can simulate on clusters, and as for the TCP/IP version, enables IP protection by hosting FMUs on secured computers.

9 Chains of mass-spring-dampers

The purpose of this experiment is to see at which point the time scales of the models and those of the couplings are sufficiently far apart that the dynamics of interest is negligibly disturbed and from there, made an estimate of the kind of time step required, in proportion to that one would use for the individual systems.

We consider a chain of N elements with ideal springs as

DOI 10.3384/ecp17132653

seen in Fig. 4

$$\bar{m}^{(j)}\ddot{\bar{x}}^{(j)} = -k^{(j-1)}(\bar{x}^{(j)} - \bar{x}^{(j-1)}) - k^{(j)}(\bar{x}^{(j)} - \bar{x}^{(j+1)}), \tag{15}$$

with $j = 2, 3, \ldots, N-1$, and

$$\begin{aligned} \bar{m}^{(1)}\ddot{\bar{x}}^{(1)} &= -k^{(1)}(\bar{x}^{(1)} - \bar{x}^{(2)}) \text{ and} \\ \bar{m}^{(N)}\ddot{\bar{x}}^{(N)} &= -k^{(N-1)}(\bar{x}^{(N)} - \bar{x}^{(N-1)}). \end{aligned} \tag{16}$$

We call these spring constants the *design* variables as they are the ones included in the original model. The undamped case is the only irrelevant one for this analysis as it is the worst case scenario for testing the schemes. This considerably reduces the dimension of the parameter space. Then we split each mass except the first and last so that for $j = 2, 3, \ldots, N-2$

$$\begin{aligned} m^{(i)} = \bar{m}^{(1)}, m^{(2N-2)} = \bar{m}^{(N)}, \text{ and} \\ m^{(2j)} + m^{(2j+1)} = \bar{m}^{(j+1)}. \end{aligned} \tag{17}$$

The interaction between $m^{(2j-1)}$ and $m^{(2j)}$ is the same as that between $m^{(j)}$ and $m^{(j+1)}$, but we now introduce spring-damper coupling $k^{(c)}, \gamma^{(c)}$ between $m^{(2j)}$ and $m^{(2j+1)}$ so that

$$\begin{aligned} m^{(2j)}\ddot{x}^{(2j)} &= -k^{(j)}(x^{(2j)} - x^{(2j-1)}) + f^{(2j,2j+1)}, \text{ and} \\ m^{(2j+1)}\ddot{x}^{(2j+1)} &= -k^{(j)}(x^{(2j+1)} - x^{(2j+2)}) - f^{(2j,2j+1)}, \\ m^{(1)}\ddot{x}^{(1)} &= -k^{(1)}(x^{(1)} - x^{(2)}), \\ m^{(N)}\ddot{x}^{(N)} &= -k^{(N)}(x^{(N)} - x^{(N-1)}), \text{ and} \\ f^{(2j,2j+1)} &= -k^{(c)}(x^{(2j)} - x^{(2j+1)}) \\ &\quad - \gamma^{(c)}(\dot{x}^{(2j)} - \dot{x}^{(2j+1)}), j = 1, 2, \ldots N-1. \end{aligned} \tag{18}$$

and therefore, we should have

$$x^{(2j)} \to_{k^{(c)} \to \infty} x^{(2j+1)}. \tag{19}$$

if the damping is correctly adjusted. We chose the non-dimensional damping parameter such that

$$\zeta = \frac{\gamma^{(d)}}{2\sqrt{\mu k^{(c)}}} = 0.7 \text{ where } \mu = \frac{m^{(2j)} m^{(2j+1)}}{m^{(2j)} + m^{(2j+1)}}. \tag{20}$$

This means that $\gamma^{(c)} \to \infty$ as $k^{(c)} \to \infty$ as required for convergence. The effect of this is also that in the stability analysis, we are at the same location in the complex plane as long as

$$h_1 \omega_1^{(c)} = h_2 \omega_2^{(c)}, \text{ where } \omega_i^{(c)} = \sqrt{\frac{k_i^{(c)}}{\mu}}. \tag{21}$$

Note that it is not generally possible to pick the damping constant γ optimally since internal inertiae and frequencies of any given FMU cannot be assumed to be known. We expect the time step to decreases linearly with the smallest period of the system, which should be $O(k^{(c)})$ unless an implicit integration strategy can be implemented.

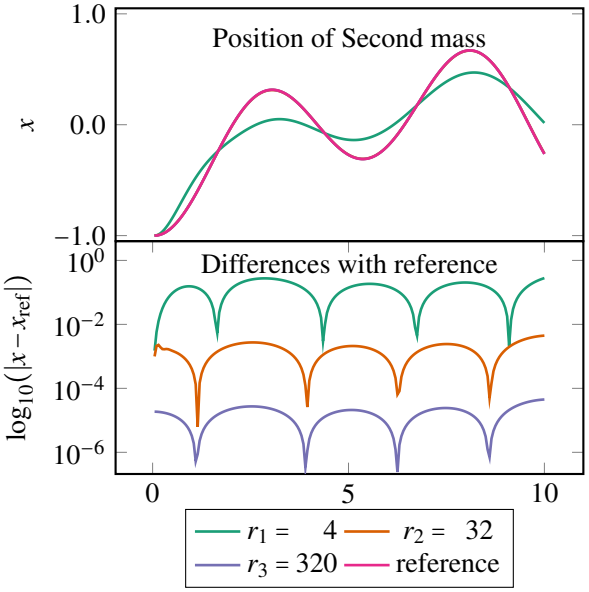

Figure 5. Influence of coupling springs on natural dynamics

We now look at ratio ρ between the coupling frequencies $\omega^{(c)}$ and those of the original system $\omega^{(d)}$, and estimate how large ρ must be to minimize interference. In our experiments, we set $x^{(1)}(0) = 1$ so as to inject energy in the natural modes. As expected for a linear system, the modes are separated and do not interact very much so the overall dynamics is similar. What is worrisome however that it takes a ratio of coupling of more than 100 before the errors go below 10^{-3}.

As seen in Fig. 4, one needs frequency ratios of around 30 to start recovering the correct solution and from Fig. 5 there is quadratic convergence towards the original solution. But given that this is a forward Euler technique, we get less. We found that we needed at least 50 times more step per *coupling* period than the minimum required by an good numerical integrator to have stability and some stability. This means $30 \cdot 50 = 1{,}500$ more steps per unit time than for the isolated models. That's three orders of magnitude more work. The DAE stepper of Sec. 6 produced very good solutions with a step commensurate with the design frequencies. We used a holonomic coupling here and included the positions in the model.

10 Experiments with a truck model

Here we investigate a simple truck model with an engine modeled with a point mass – the flywheel –, a PI control which aims at reaching a given speed, a clutch, a gearbox and a shaft, each represented with a pair of masses coupled with spring-dampers – piecewise linear for the clutch as in Fig. 7–, and a trailer modeled as a point mass but interacting with a road with variable slope following a sine

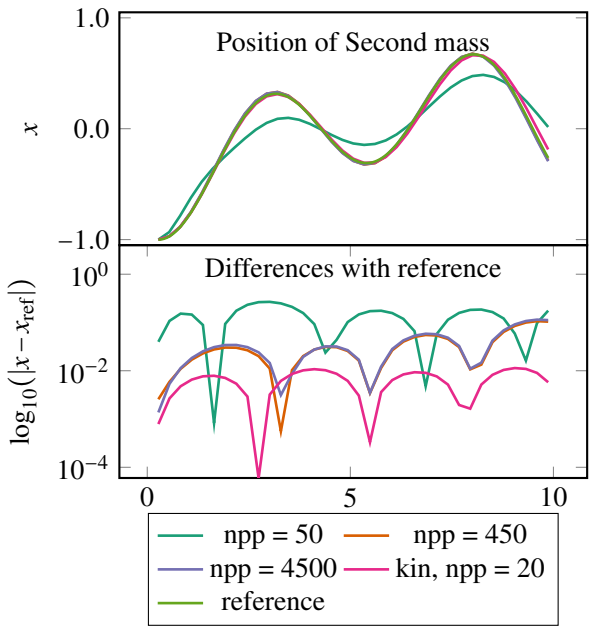

Figure 6. Simulation results for chains

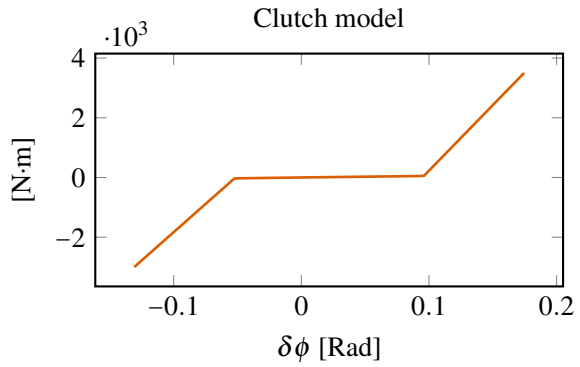

Figure 7. A piecewise linear clutch model

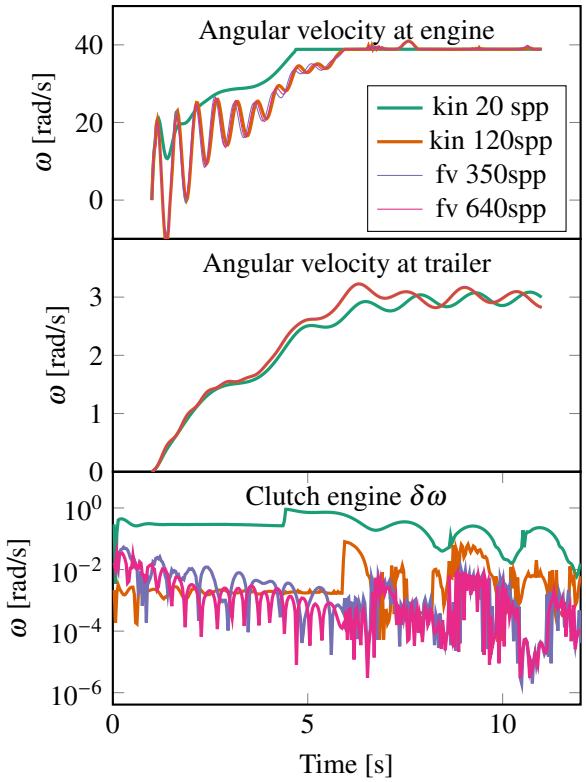

Figure 8. Cosimulation of a truck model. The "spp" key stands for steps per period.

wave, and subject to gravity, rolling and dry friction, as well as air resistance. This is a textbook model (Eriksen and Nielsen, 2014).

The engine delivers 1,350 Nm max torque, the target speed is 100 km/h, and the trailer has a mass of 10,000 kg. The slope of the road was made sinusoidal. The interesting aspects here are the mass ratio and the large torques involved. The kinematically coupled model simply constraints velocities and coordinates between the components, i.e., the flywheel angle should match that of the in plate of the clutch, etc.

Each FMU has it's own time integration and we chose the GSL for that. We used the fourth order Runge Kutta method `rkf45` for our experiments with 10^{-6} tolerance. For the kinematic stepper, we computed the effective mobility by integrating and rollback and then using finite differences.

Here we compare our kinematic stepper of Sec. 6 with a step of $1/20$ and $1/120$ of the smallest period in the design. In this case, this is in the clutch and gearbox dynamics. The case of 120 steps per period offers good results, though as low as 10 steps per period as stable. However, for the force-velocity coupling using either sequential or parallel simulation required more than 350 steps per period before stability. Good results come after 640 steps per period. Things were worse yet when we used holonomic coupling, i.e., including spring dampers for positions as well as velocities. We needed more than 10 times as many steps for the force-velocity versions, though with kinematic coupling, we had high accuracy at 20 steps per period already (results not shown). To be considered here is that the mass of the trailer is so much larger than the driveline that very stiff springs would be needed to reach the correct result. Considering the previous experiment in Sec. 9 we used coupling springs 30 times larger those in the design. This leads to $30 \cdot 350/20 \approx 500$ more steps than necessary just for stability. The DAE stepper does perform more work per step: solving a small system of linear equations, computing directional derivatives, and performing two sub-steps per step. But even that's not a fair comparison since the FMUs in the kinematic coupling setup do not contain stiff coupling springs and therefore, they also perform an order of magnitude less work. In this case, the best result from the DAE stepper used four times fewer integration steps overall, and that's despite the fact that we used numerical directional derivatives, which takes four times as many steps.

DOI 10.3384/ecp17132653

11 Discussion

The main lesson here is that when it comes to force-velocity couplings at least, but this applies to the other cases of spring-damper type couplings, one needs at least 30:1 ratio of frequencies and as far as ZOH techniques goes, this introduces orders of magnitude more work than an all-at-once method, or a DAE based one. Though the frequency ratio appears inevitable, there is recent work (Drenth, 2016) indicating that the communication step size can be kept modest by filtering the high oscillations. We have not been able to use this technique or reproduce the results in our experiments, something which the author relates to admittance or mobility ratio between the FMUs.

Kinematic coupling offers very good solutions at relatively large steps and were able to use holonomic couplings as well at moderate steps. Of course, this requires functionality that is not often seen in CS FMUs, namely, rollback and directional derivatives.

12 Conclusion

The software we introduce should be of interest for being minimalistic and offering a good foundation to build integration environments for cosimulation. It offers very good performance for a standalone computer yet can be distributed over WAN. We hope that the functionality we are developing will open the door to more advanced time integration methods. We will soon start collaborations with the OpenModelica and OpenCPS groups, which will provide user interfaces.

Our kinematic stepper can produce very good results in keeping time steps commensurate with the model frequencies and offers parallelism, at least on simple models. Further investigation is needed clearly, but the benefit in comparison to force-velocity coupling is significant and we believe that, despite the difficulties associated with roll-back and computation of directional derivatives, is worth much attention.

As part of future work, we intend to support TLM as it is popular, provide a fully functional ME simulation master, and improve the numerical directional derivatives functionality to be fully parallel. Other features such as extrapolation and interpolation in the ME FMU wrapper, as well as iterative and implicit time integration methods, or various types of filtering are under consideration.

The software is available on a request basis at this time at *git clone* at `https://mimmi.math.umu.se/users/sign_in`. Anonymous access is forthcoming.

Acknowledgments

This work is a part of the project "Virtual Truck and Bus" supported by the Swedish Energy Authority, and is a collaboration between Scania CV AB, Algoryx Simulation AB, Modelon AB, Umeå University and Volvo Car Corporation. Previous contributions to the software development were made by Stefan Hedman and Adeel Ashgar.

Bibliographic References

References

Arnold, Martin (2010). "Stability of Sequential Modular Time Integration Methods for Coupled Multibody System Models". In: *Journal of Computational and Nonlinear Dynamics* 5.3, pp. 031003–031003.

Ascher, Uri M. and Linda R. Petzold (1993). "Stability of Computational Methods for Constrained Dynamics Systems". In: *SIAM J. Sci. Computing* 14.1, pp. 95–120.

Awais, M. U. et al. (2013). "Distributed hybrid simulation using the HLA and the Functional Mock-up Interface". In: *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, pp. 7564–7569.

Broman, D. et al. (2013). "Determinate composition of FMUs for co-simulation". In: *2013 Proceedings of the International Conference on Embedded Software EMSOFT*, pp. 1–12.

Cremona, F. et al. (2016). "Step revision in hybrid Co-simulation with FMI". In: *2016 ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, pp. 173–183.

Dag Fritzson Johas Ståhl, Iakov Nakimovski (2007). "Transmission line co-simulation of rolling bearing applications". In: *The 48th Scandinavian Conference on Simulation and Modeling*. Ed. by Claus Führer Peter Bunus Dag Fritzson, pp. 24–39.

Davis, Timothy A. (2004). "Algorithm 832: UMFPACK — an Unsymmetric-Pattern Multifrontal Method". In: *ACM Transactions on Mathematical Software* 30.2, pp. 196–199.

Drenth, Edo (2016). "Robust Co-Simulation Methodology of Physical Systems". In: *9th Graz Symposium Virtual Vehicle*.

Eriksen, Lars and Lars Nielsen (2014). *Modeling and control of engines and drivelines*. John Wiley & Sons.

Fiedler, Robert and Martin Arnold (2014). "Coupled differential algebraic equations in the simulation of flexible multibody systems with hydrodynamic force elements". In: *PAMM* 14.1, pp. 523–524.

Galtier, Virginie et al. (2015). "FMI-based Distributed Multi-simulation with DACCOSIM". In: *Proceedings of the Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. DEVS '15. Alexandria, Virginia: Society for Computer Simulation International, pp. 39–46.

Protocol Buffers (2017). `https://github.com/google/protobuf`.

Hairer, E. (2000). "Symmetric Projection Methods for Differential Equations on Manifolds". In: *BIT Numerical Mathematics* 40 (4), pp. 726–734.

Hairer, E., C. Lubich, and G. Wanner (2001). *Geometric Numerical Integration*. Vol. 31. Springer Series in Computational Mathematics. Berlin: Springer-Verlag.

IEEE (2010). “IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules”. In: *IEEE Std 1516-2010*, pp. 1–38.

iMatrix (2017). *ZeroMQ*. http://zeromq.org/.

Köler, Jochen et al. (2016). “Modelica-Association-Project “System Structure and Parametrization" – Early Insights”. In: *Proceedings of the 1st Japanese Modelica Conference*. Modelica Association. Linköping University Electronic Press, pp. 35–42.

Krus, Petter (1995). “Modelling of Mechanical Systems using Rigid Bodies and Transmission Line Joints”. In: *ASME J. Dyn. Sys., Meas., Control* 121.4, pp. 606–611.

Lacoursière, Claude (2007). “Ghosts and Machines: Regularized Variational Methods for Interactive Simulations of Multibodies with Dry Frictional Contacts”. PhD thesis. Dept. of Computing Science, Umeå University.

Lacoursière, Claude and Sjöström (2014). *A non-smooth event-driven, accurate, adaptive time stepper for simulating switching electronic circuits.* Tech. rep. UMINF 16.15. Dept. of Computing Science, Umeå University.

M. Benedikt et al. (2013). “NEPCE - A nearly energy-preserving coupling element for weak-coupled problems and co-simulations”. In: *V International Conference on Computational Methods for Coupled Problems in Science and Engineering*. Ed. by S. Idelsohn, M. Papadrakakis, and B. Schrefler, pp. 1021–1032.

Martin, Arnold, Clauss Christoph, and Schierz Tom (2013). “Error Analysis and Error Estimates for Co-Simulation in FMI for Model Exchange and Co-Simulation V2.0”. In: *Archives of Mechanical Engineering* 60. 1, pp. 75–94.

Microsoft (2017). *Microsoft MPI*. http://tinyurl.com/mpilib-microsoftv85.

MIT (n.d.). *MIT license*.

MODELISAR (2014). *FMI website*. last retrieved 2017-01-22. URL: https://www.fmi-standard.org.

MPI (2017). *A Message Passing Interface Standard*. http://mpi-forum.org/.

MPICH (2017). *High performance, widely portable implementation of the Message Passing Interface*. http://mpi-forum.org/.

Ptolemaeus, Claudius, ed. (2014). *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org. URL: http://ptolemy.org/books/Systems.

QTronic (2017). *QTronic FMI SDK*. http://www.qtronic.de/en/fmusdk.html.

Schierz, Tom and Martin Arnold (2012). “Stabilized overlapping modular time integration of coupled differential-algebraic equations”. In: *Applied Numerical Mathematics* 62.10. Selected Papers from NUMDIFF-12, pp. 1491–1502.

Schierz, Tom, Martin Arnold, and Cristoph Clauss (2012). “Co-simulation with communication step size control in an FMI compatible master algorithm”. In: *Proceedings of the 9th International MODELICA Conference*.

Schweizer, B. and D. Lu (2015). “Predictor/corrector co-simulation approaches for solver coupling with algebraic constraints”. In: *ZAMM* 95 (9), pp. 911–938.

Schweizer, Bernhard and Pu Li (2015). “Solving Differential-Algebraic Equation Systems: Alternative Index-2 and Index-1 Approaches for Constrained Mechanical Systems”. In: *Journal of Computational and Nonlinear Dynamics* 11.4, pp. 044501–044501.

Schweizer, Bernhard, Pu Li, and Daixing Lu (2015). “Explicit and Implicit Cosimulation Methods: Stability and Convergence Analysis for Different Solver Coupling Approaches”. In: *Journal of Computational and Nonlinear Dynamics* 10.5, pp. 051007–051007.

Schweizer, Bernhard, Pu Li, Daixing Lu, and Tobias Meyer (2015). “Stabilized Implicit Cosimulation Method: Solver Coupling With Algebraic Constraints for Multibody Systems”. In: *Journal of Computational and Nonlinear Dynamics* 11.2, pp. 021002–021002.

Schweizer, Bernhard, Daixing Lu, and Pu Li (2015). “Co-simulation method for solver coupling with algebraic constraints incorporating relaxation techniques”. English. In: *Multibody System Dynamics*, pp. 1–36.

Sjölund, Martin et al. (2010). “Towards Efficient Distributed Simulation in Modelica using Transmission Line Modeling”. In: *Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Languages and tools*. Ed. by Peter Fritzson et al., pp. 71–80.

Zeigler, Bernard P., Herbert Praehofer, and Tag G. Kim (2000). *Theory of Modeling and Simulation*. 2nd ed. Academic Press.