General meta-model based co-simulations applied to mechanical systems

Alexander Siemers a,*, Dag Fritzson b,1, Iakov Nakhimovski b,2

a Linköpings Universitet, Institutionen för Datavetenskap, 58183 Linköping, Sweden
b SKF Engineering and Research Centre, MDC/Rks-2, 41550 Gothenburg, Sweden

A B S T R A C T

A fully functional meta-model co-simulation environment that supports integration of many different simulation tool specific models into a co-simulation is described in this paper.

The continuously increasing performance of modern computer systems has a large influence on simulation technologies. It results in more and more detailed simulation models. Different simulation models typically focus on different parts (sub-systems) of the complete system, e.g., the gearbox of a car, the driveline, or even a single bearing inside the gearbox. To fully understand the complete system it is necessary to investigate several or all parts simultaneously. This is especially true for transient (dynamic) simulation models with several interconnected parts. One solution for a more complete and accurate system analysis is to couple different simulation models into one coherent simulation, also called a co-simulation. This also allows existing simulation models to be reused and preserves the investment in these models.

Existing co-simulation applications are often capable of interconnecting two specific simulators where a unique interface between these tools is defined. However, a more general solution is needed to make co-simulation modelling applicable for a wider range of tools. Any such solution must also be numerically stable and easy to use in order to be functional for a larger group of people.

The presented approach for mechanical system co-simulations is based upon a general framework for co-simulation and meta-modelling [9]. Several tool specific simulation models can be integrated and connected by means of a meta-model. A platform independent, centralised, meta-model simulator is presented that executes and monitors the co-simulation. All simulation tools that participate in the co-simulation implement a single, well defined, external interface that is based on a numerically stable method for force/moment interaction.

© 2008 Published by Elsevier B.V.

1. Introduction

In the area of modelling and simulation of mechanical systems one can identify many different classes of models and corresponding tools, each optimised for a certain kind of analysis. In terms of meta-modelling, every tool can be seen as a black box handling a particular external model. An external model is a model defined in some specific modelling language supported by modelling and simulation tools that can perform a simulation of it. Examples of such models are equation based multiphysics Modelica [8] models, multibody models in MSC.ADAMS [18], models with detailed contact definitions in SKF's
BEAST [4], control system models in Simulink, flexible models as modelled in Finite Element (FE) tools, etc. In reality, the different sub-systems represented by these models are often physically tightly coupled and interdependent. The need to couple different models into a complete more tightly integrated model that can be simulated is therefore justified. Furthermore, this allows preserving the investments in these models.

Co-simulation is one technique for coupling different simulators into one coherent simulation. Typically, co-simulation applications are capable of interconnecting two or more specific simulators where an interface between these tools is defined. To make co-simulation applicable for a larger range of tools more general solutions are needed. Any such solution must be adaptable to many existing simulation tools, easy to use from a user perspective, and numerically stable. The proposed solution described in this paper is to integrate and connect several tool specific simulation models by means of a meta-model.

Meta-models allow for centralised simulation control where a single simulation manager application uses the meta-model to execute and control the co-simulation. All communicated data can be monitored for later analysis. The meta-model forms a platform independent representation of the total model controlling the co-simulations.

An approach for co-simulation meta-modelling has been presented in [9]. In this earlier work the Modelica language [8] is used to describe the meta-model in an easy to understand, object oriented way. A ModelicaXML based translator is used to convert Modelica code into an XML document that is accepted as input by the co-simulation environment. Afterwards the modelling and simulation environment has been extended to better support meta-modelling for mechanical system simulations, as described in [10,12]. In this paper, the system design of the meta-model simulation environment is presented.

Different co-simulation environments have appeared during recent years. Most of them are focused on co-simulation of control systems and corresponding mechanical components, hardware software co-simulation, or embedded system simulation, e.g., Cosimate [17], Ptolemy-II [16], MILAN [6], and the integrated co-simulation environment for heterogeneous systems prototyping [3]. MILAN is a model based, extensible environment that integrates different widely used embedded system simulators into a unified environment. Its user interface is based on the generic modelling environment [7] (GME) and allows for hierarchical data flow modelling. Ptolemy II supports actor oriented design where actors communicate by message passing through well defined communication channels. This contrasts with method invocation in object oriented design. One of Ptolemy II’s strength is domain polymorphism meaning that simulation components are designed to operate in different domains. Ptolemy II models are hierarchical data flow models where data flow defines the communication channels between the actors. In contrast to MILAN and Ptolemy II stands Cosimate which avails itself to a bus architecture. All co-simulation components communicate through the co-simulation bus. This allows for monitoring of bus signals and thus for data-flow control. The simulations performed by these tools as typically discrete time simulations. The coupled simulations in the work presented here are different. Most important, from a numerical point of view, the co-simulation components are likely to use different differential equation solvers possibly with variable time steps. Numerical stability, which is not an issue for discrete time simulations, becomes an important consideration.

Besides, the meta-model environment presented here focuses on mechanical system modelling and introduces uniform interfaces that allow for the stable interconnection of various simulation tools and numerical solvers. Meta-models also represent portable entities that are operating system, network infrastructure, and computer system independent.

2. Meta-models

In the context of this work a meta-model defines the physical interconnections of various external models, where an external model is a model defined in some specific language supported by modelling and simulation tools that can perform a simulation of it. Each external model has at least one, but can have several external interfaces for connection with other external models. By interconnecting several external models the (extended) overall structure of the system model is created. The meta-model defines the semantics of each external model and their interconnections within this structure, see Fig. 1.

![Fig. 1. The meta-model defines the interconnections between various external models.](image-url)
Meta-models, as defined here, provide:

- A language representation with strict notation and grammar. This is used to represent and store the meta-model and to share meta-model information between different tools and people.
- Hierarchical modelling, where larger systems can be modelled using various sub-models. Sub-models are intermediate level nodes in the meta-model and define different hierarchical sub-levels, each of which might contain external models, connections between the external models, and other sub-models.
- Model abstraction, where each external model is represented individually in a uniform and simulation tool independent way.
- A generic and uniform way to connect various simulation tools.
- Platform independent models. No operating system, network, or other co-simulation platform dependent parameters are stored in the meta-model.
- Graphical language elements for visual representations of the external models.
- Centralised co-simulation control. A single meta-model simulator application can start the simulation tools and control the co-simulation based on the meta-model.

A meta-modelling process has been defined that forms the basis of this work. The process can be divided into three steps:

- **External model design** in a specialised environment. Each external model in the co-simulation is modelled separately in its specific environment.
- **Model integration** into the meta-modelling environment. First of all the external interfaces are defined in the external models. Subsequently, all external models need to be encapsulated for integration into the meta-model. Startup methods, interface names, and possibly geometry data must be specified.
- **Meta-model design** in a meta-model editor. The different external models are integrated into the meta-model and connect with each other. Global simulation parameters also need to be defined.

There are different levels of expertise required in the different phases of the modelling process. All work in a specialised simulation environment requires a high level of expertise. On the other hand, the meta-model design phase requires the lowest level of expertise and is based on a uniform meta-model editor.

As an example, the modelling of a vehicle is considered, see Fig. 2. A car with an integrated hub unit has been modelled and simulated to verify the hub unit design. Here MSC.ADAMS [18] performs the simulation of a car with suspension and wheels. A detailed model of the hub unit including the bearings is simulated in BEAST [4]. The meta-model contains two external models, one for the car body and one for the hub unit. Both external models contain two external interfaces that are connected in the meta-model to mount the front left wheel with the hub unit to the car suspension. Physical connection properties and global simulation parameters, e.g., simulation start and end time, are also defined in the meta-model.
2.1. Meta-model language

The meta-model language (MML) is used to define meta-model structure in a flexible way. The first version of the meta-model language is based on an XML [9], which has subsequently been redesigned to meet the following requirements:

- Remove XML tags to improve readability
- Add hierarchical constructs to allow for hierarchical meta-models
- Allow for geometrical elements for improved meta-model visualisation

Meta-models are organised in a hierarchical fashion building the meta-model tree. Language elements are directly related to meta-model tree nodes. These are classes of objects (here called categories) each containing data and functionality in the form of semantic definitions.

The language defines strict rules for hierarchical composition, model interfaces, naming conventions, and element attributes and parameters. In particular the following categories and hierarchical composition rules exist:

- **Environment** is the highest node in the meta-model. It contains the meta-model's inertial system and the root model. Global meta-model simulation parameters are specified here, these are: model name, simulation start time, and simulation end time.
- **Models** are used as intermediate level nodes in the meta-model. Models define different hierarchical sub-levels. They contain other Models, XModels and Connections.
- **XModels** are the external models. They contain: information about the simulation files and startup methods, a constant coordinate transformation, one or more control points, and XGeometry.
- **Connections** connect two XModels exactly. A connection is a container for all the Ties between any two XModels.
- **Ties** represent the physical interconnections in the meta-model. Each tie connects two control points exactly. They store connection parameters used by the manager of the meta-model simulation.
- **Coordinate systems**, also called control points, are the interfaces for communication in a meta-model simulation and represent the external interfaces. They contain an interface name and information about the motion of the interfaces coordinate system relative to the external models local inertial system.
- **Implicit coordinate systems** define the meta-models global inertial system and the external model's local inertial systems.
- **XGeometry** contains the external models surface geometry for 3D visualisation.

A meta-model is mapped to a directory structure. Each external model represents an encapsulated entity where all files, i.e., simulation input files and possible geometry (VRML) files, are gathered into a single directory. The directory is named after the XModel. Encapsulation improves re-usability, i.e., the same external model can be used multiple times in the same meta-model without changing simulation input file names, and allows for easy data distribution, i.e., complete directories can easily be packed, copied, and distributed (including simulation output files).

Here is an outline of the MML file for the "car with hub" model, defining the XModel of the car with geometry and external interfaces:
environment
def time=0 unit="s";
def StartWriteTime=0 unit="s";
def timeEnd=2.5 unit="s";
model type=SModel name=""
xmodel type=XModel name=AdamsCar
def StartCommand="StartTLMAdams";
def SimulationFile="CarTLM.acf";
def ExtraSimFiles="CarTLM.bin,
CarTLM.cmd";
def cX'R'[cG][cG]=0, 0, 0 unit="m";
def cX'phi[cG]=0, 0, 0 unit="rad";
def cX'phiSequence[cG]=0;
coordinate_system type=XCtlPoint
  name=ctTLMknuckle
def TLMPortName="M17305495";
def R[cX]={[0,
  0,7342,
  0,2624]}
  unit="m";
def phiSequence[cX]=0;
def phi[cX]={[5.1415926536,
  1.5707963268,
  1.5707963268]}
  unit="rad";
end coordinate_system;
...
xgeometry type=XGeometry name=xsl
  def CADFileName="Car.wrl";
end xgeometry;
end xmodel;
...
connection type=XModelXModelConnection
  from=AdamsCar to=xmHubUnit
tie type=TLMTie
  from=ctTLMknuckle to=ctbER'ctl2
def delay=2.0e-5;
def Zf=1.0e4;
def Zfr=100;
def alpha=0.01;
end tie;
...
end connection;
end model;
end environment;

2.2. Meta-model editor

In earlier work meta-models were designed using a graphical Modelica editor and stored as Modelica models [9]. This work has been continued to improve meta-model simulation pre- and post-processing.
The meta-model editor (MME) features three dimensional representations of all meta-model components, see Fig. 3, plus a hierarchal view of the complete model tree. External models can be imported into the meta-model including graphics for visual representation. The editor fully supports the meta-model paradigm and is independent of any simulation tool. The editor thus supports both model integration and meta-model design. MME directly writes the MML file and organises the meta-model directory structure.

3. Co-simulation framework

A general framework for meta-model based co-simulations has previously been designed [9]. This framework has been extended to support: inertial coordinate system alignment between different external models, monitoring of all communication between the external model simulations, and a new meta-model language (MML).
The design goals for the simulation part of the framework were portability, simplicity to incorporate new simulation tools, and computational efficiency. These goals were realized by defining the following concepts and interfaces:

### 3.1. External interface

A named point on a mechanical object where position and velocity can be evaluated and reaction load (force and moment) applied. To guarantee numerical stability when utilising different numerical solvers in the co-simulation, only interfaces based on the transmission line modelling method, see Section 4, are currently supported.

### 3.2. Simulation manager

The central simulation engine. It is a stand alone program that reads an MML definition of the coupled simulation. It then starts external model simulations and provides the communication bridge between the running simulations. The external models only communicate with the simulation manager which acts as a broker marshalling information between the external models. Simulation manager sees every external model as a black box having one or several external interfaces. The information is then forwarded between external interfaces belonging to different external models. Additionally the simulation manager opens a network port for monitoring all communicated data.

### 3.3. Interface plug-in

A small C++ library having a single abstract class representing external interface for a specific simulation tool. The interface plug-in can be seen by an external model simulator as an external load that depends on position, velocity and time. The implementation of the plug-in handles the necessary communications with the simulation manager. It also handles necessary coordinate system transformations into the global meta-model inertial system. All positions and velocities are transformed from the local (external model) inertial system to the global inertial system. All reaction loads are translated back into the local inertial system. This constant transformation is stored in the meta-model and sent to the corresponding interface plug-in on simulation start up.

### 3.4. External model simulator

Any simulation program that has incorporated the interface plug-in as a part of its model. A small script that takes the general parameters as input and starts the specific simulation tool is an additional requirement. This intermediate step is necessary since the simulation manager needs a common way to start all the components and each tool might have some specific start procedures.

### 3.5. Requirements on external model simulators

External models are associated with specialised simulation tools. Even though many simulation tools have interfaces to external functions, the interfaces differ between tools. Therefore, it is first necessary that a software developer who is familiar with the particular tool architecture, designs and implements the external interface for each tool. That is, to create a tool specific wrapper for the interface plug-in of the simulation framework.

The following functionality is required from all simulation tools that implement the interface plug-in and want to participate in co-simulation:

- Possibility to start simulation externally or in batch mode. The developer of the tool specific interface must provide a start up program.
- Possibility to integrate the interface plug-in into the tool specific adaptor. Note that the tool independent part of the plug-in is implemented in C++. Some tools require external functions to be implemented in, e.g., C or Fortran. In such cases the C++ code can be invoked from a C or Fortran function.
- Ability to deliver position, orientation and velocity of a point to the interface plug-in and receive the reaction load (force and moment) to be applied at this point.
- Correct handling of shutdown signals coming from the tool specific wrapper. In some cases the simulation manager needs to take down the simulation tool in a controlled way. This can be achieved by a tool specific API (Application Programming Interface) call or simply handling of exit signals.

Additionally, it is desirable that the simulation tool can export surface geometry (graphics) for visualisation in the meta-model environment. Surface geometry is not required for correct meta-modelling but beneficial for visual model verification.

Most of the commonly used simulation tools offer some kind of external connection either through inter process communication (IPC), e.g., network sockets or remote procedure calls, or an application programming interface (API). Both options are acceptable for implementation of the interface plug-in as long as they fulfil the requirements above. The main focus of this work is on transient simulations of mechanical systems. Simulation tools that are of interest for this work are pure
mechanical system and multi physics tools. All of the tools that have been considered for integration into the co-simulation framework comply with the requirements, some of the tools are shown in Table 1. Interface plug-ins for SKF’s BEAST, SKF’s Orpheus, MSC.ADAMS, Matlab/Simulink, and Modelica have successfully been implemented and tested.

4. Transmission line based co-simulations

The method that is used to enable interaction between dynamic models in the meta-model simulation is transmission line modelling (TLM) [1,5,2,11]. TLM uses physically motivated time delays to separate the components in time and enable efficient co-simulation. Only TLM connections between two external interfaces are currently supported by the meta-model simulation environment because the TLM method gives numerical stability.

4.1. TLM background theory

The TLM method, also called Bilateral Delay Line Method [1], exploits the fact that all physical interactions have finite propagation speed.

A basic one dimensional transmission line is shown in Fig. 4. For the mechanical case the line is basically an ideal elastic medium with force waves \( c_1 \) and \( c_2 \) going between its ends. The input disturbances are velocities \( v_1 \) and \( v_2 \) and the forces from the transmission line \( F_1 \) and \( F_2 \).

Note that the springs in our implementation are assumed to be isotropic, i.e., no cross-term waves are generated when working in 2D and 3D. See [5] for further discussions.

If the line delay is set to \( T_{TLM} \) and its impedance to \( Z_F \) then the governing equations are:

\[
\begin{align*}
    c_1(t) &= F_2(t - T_{TLM}) + Z_F v_2(t - T_{TLM}) \\
    c_2(t) &= F_1(t - T_{TLM}) + Z_F v_1(t - T_{TLM}) \\
    F_1(t) &= Z_F v_1(t) + c_1(t) \\
    F_2(t) &= Z_F v_2(t) + c_2(t)
\end{align*}
\]

The equations show that the two simulation systems are decoupled with the delay time \( T_{TLM} \). Simulation framework can utilise this decoupling to enable efficient communications during co-simulation.

Representing the TLM connection with a simple model of a steel beam, the stiffness coefficient can be computed as (see [5]):

\[
k = \frac{EA}{L_0}
\]

where \( E \) is Young’s modulus, \( A \) is the cross section area and \( L_0 \) the length of the beam.

The impedance \( Z_F \) has a relation to the spring constant \( k \), \( Z_F kT_{TLM} \). The impedance factor can then be formulated as a function of the area and length of the steel rod according to:

\[
Z_F = \frac{EA T_{TLM}}{L_0}
\]

To get the stiffness and impedance for the rotational degrees of freedom one can use the already computed stiffness \( k \). If the arrangement depicted in Fig. 5 is assumed, then:

![Fig. 4. Delay line with the passing wave variables \( c_1 \) and \( c_2 \) and velocity variables \( v_1 \) and \( v_2 \).](image-url)

### Table 1

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Implementation</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEAST (SKF in-house)</td>
<td>C++ implementation</td>
<td>TLM enabled control points (coordinate-systems)</td>
</tr>
<tr>
<td>MSC.ADAMS</td>
<td>C wrapper DLL (dynamic link library)</td>
<td>General force with sub-routine call</td>
</tr>
<tr>
<td>Matlab/Simulink</td>
<td>C wrapper</td>
<td>S-function interface</td>
</tr>
<tr>
<td>Modelica</td>
<td>C or Fortran wrapper</td>
<td>External function interface</td>
</tr>
<tr>
<td>Simpack</td>
<td>Fortran wrapper</td>
<td>SIMPACK User routine</td>
</tr>
</tbody>
</table>

The list of potential simulators considered for TLM co-simulation. Possible implementation and type of the interface plug-in are also shown.
\[
\begin{align*}
k_\phi &= M \left( \frac{k}{2} \right)^2 \frac{\delta_\phi (L_0/2)^2}{\delta_\phi} \frac{k L_0^2}{4} \\
Z_{FR} &= \frac{1}{4} Z_L L_0^2 \\
T_{TLM} &= \frac{L_0}{v_{\text{medium}}} \\
m_p &= Z_f T_{TLM}
\end{align*}
\]

Fig. 5. Estimating the rotational stiffness.

The most obvious reason for using TLM is that it provides a structured and stable way to connect to simulators in a co-simulation set up. The TLM element does not introduce any numerical problems for the solvers. The price to pay is that a new component has to be introduced in the model. This component behaves as a medium for the communicating “wave” between the simulated processes. Therefore, this adds a model error, but that error is always explicit and well defined compared to errors introduced “under the hood” due to numerical issues in the simulations.

In order to avoid the impact of the model error from the TLM element it is always important to let the TLM part represent a natural elasticity in the total system. Then the TLM element can be tuned to adapt (or model) that elasticity as much as possible, making the model error as small as possible.

5. Meta-model simulation

A meta-model simulation environment has been created that is based on the previously defined co-simulation framework. The system design of the environment is shown in Fig. 6. The environment provides:

- Generality, due to the general framework that allows for integration of many different simulation tools.
- A generic method of co-simulation based on a meta-model.
- A general method of external model execution, i.e., all simulation tools involved are executed. Generality is achieved by a platform independent start up command that takes care of possible actions, i.e., remote login, file transfer, or set up of a parallel simulation environment.
• Communication and data transfer between the different external models.
• Data monitoring for analysis and post processing.
• Controlled simulation termination where all external models are taken down in a correct way. This includes error handling due to external model failure or network problems.

In addition to the simulation manager the meta-model simulator has been designed that starts and monitors the meta-model simulation. It creates its own internal meta-model structure for best simulation control and storage of meta-model simulation results. The simulation manager has been designed to act as a network server that allows several clients to connect and monitor the data flow between the interconnected external model simulators. By default the simulator connects to this port. All data received by the simulation manager from the external model simulators is passed on to the simulator that stores results in the meta-model output file. This client server architecture has some advantages over an integrated solution where the simulation manager is part of the simulator:

• The simulator and simulation manager can run on different machines, this is useful, for instance, if the simulation start up must be performed on a machine other than the user’s local machine.

---

Fig. 6. The system design for the meta-model simulation environment.

Fig. 7. The simulation manager handles start up and communication between the tool specific simulators.
Other processes can connect to the simulation manager’s monitoring port. This allows, for instance, to visually monitor the co-simulation.

The simulation manager manages start up of the tool specific simulation tools (external model simulators) and communication between these tools, as defined by the simulation framework, see Fig. 7. Each simulation tool implements the interface plug-in that handles the necessary communication with the simulation manager. The manager keeps an internal interconnection table for all connected external interfaces and forwards all received packages accordingly.

Simulation results can be analysed in the post-processing applications, i.e., a two dimensional plotting program and a three dimensional visualisation tool, i.e., the meta-model editor (MME). Data animation of system dynamics is possible to a limited extent based on the data that is exchanged in the external interfaces.

6. System verification

The TLM interfaces have been implemented and tested for SKF’s BEAST, MSC.ADAMS, Simulink, Modelica, and SKF’s Orpheus that is used for studying and optimising the dynamic behaviour of noise and vibration in critical bearing applications. Each implementation is tested in a systematic way. A pendulum with three connected shafts has been designed as the test meta-model, see Fig. 8. Each shaft is modelled as a separate external model. The shafts are connected via TLM interfaces.

The model has all important characteristics to test the simulator and various TLM interface implementations:

- Multiple simulation models to test various combinations of external models.
- Open (not connected) external interface in the endShaft. The simulation should not fail due to open external interfaces but respond with zero force and moment in the interface.
- Initial position and orientation of the midShaft and endShaft. The shaft models have different local inertial systems. For correct meta-model simulation the local inertial systems need to be aligned with the global meta-model inertial system. The necessary transformations are stored in the meta-model and aligned before any data transformation between the external models.

All shafts have the same properties in all simulation tools. Each of the shafts has a mass of 3 kg, a length of 160 mm, and a diameter of 40 mm. The test procedure for a new TLM implementation, e.g., the Simulink interface plug-in, is the following:

1. Encapsulate the simulation models and perform an interface data request. This starts the simulation of the tool specific model and sends position and orientation of all TLM interface points for the first simulation time step.
2. Create a meta-model with three shafts models from the same simulation tool, e.g., three external Simulink models. Here we test for correct interface alignment and meta-model execution. The results from the meta-model simulation are compared to the reference case, i.e., the pure BEAST meta-model.
3. Replace one or two shafts with models from other simulation tools. This is to test the implementation in conjunction with other implementations. The results are compared to the reference case, i.e., the pure BEAST meta-model.

Results from the meta-model simulation can be verified in a post-processing application. Fig. 9, for instance, shows the moments created in the TLM interfaces of three connected external Simulink models. The time delay in the connected interfaces can be seen due to TLM delay.

7. Performance evaluation

Different tests have been conducted to investigate the influence of different parameters on the simulation and co-simulation performance. Fritzson et al. [13] investigated the impact of TLM elements on the numerical performance of a simulation and the influence of different TLM parameters on the simulation results. It has been show that TLM based co-simulation...
works and performs well if the values for $T_{\text{TLM}}$ are based on physical estimates. Larger values for $T_{\text{TLM}}$ have a negative influence on the agreement of TLM versus non-TLM simulation, which is understandable since the physical properties of the overall system changes, i.e., the parasitic mass becomes larger and the different subsystems are less tightly coupled. The conclusion was that the agreement between the non-TLM and the TLM analyses was good, both in terms of physical behaviour and numerical stability, if $T_{\text{TLM}}$ is chosen carefully (based on physical estimates).

Norling et al. [14] investigated which parameters affect the total simulation time of a co-simulation on a wide area network (WAN) using the presented co-simulation environment. It has been shown that it is possible to run co-simulations over long distance networks (between Europe and Australia) with acceptable performance. For the investigation the following

![Fig. 9. The difference in Moment between the TLM connected coordinate systems due to TLM delay.](image)

(a) Moment in TLM connected coordinate systems cGShaft-midShaft. (b) Moment in TLM connected coordinate systems midShaft-endShaft.

![Fig. 10.](image)

(a) The shaft model used in the extendable meta-model. (b) The ball bearing model used in the pendulum meta-model. A simplified bearing with three balls has been used to shorten simulation time.

(c) An example of the extendable pendulum meta-model with four shafts and three ball bearings. The picture is taken from the 3D post-processor.

Fig. 10. The extendable meta-model can be assembled by connecting one to several shaft models with one to several ball bearing models.
parameters were chosen: $T_{TLM}$, networks bandwidth, and network latency. The conclusion was that $T_{TLM}$ and network latency have a large impact on the simulation performance. Larger $T_{TLM}$ values had a positive effect on the total simulation time. This is because $T_{TLM}$ has a direct influence on the maximum time step of the connected simulation tools. Larger time steps can be taken, depending on the model and the simulation tool, and the simulation time decreases. However, one should remember that $T_{TLM}$ has an influence on the physical properties of the model and thus the simulation result. It has also been shown that larger network latency had a negative effect on the simulation performance. A “bottleneck breakpoint” is described in which the influence of network latency on the total simulation time dramatically changes (from low influence to high influence). This point moved towards lower latency values when the simulation performance was increased. Which can be explained with the increasing communication frequency for increasing simulation performance. Network latency takes thus a larger fraction of the total simulation time. Maximum network bandwidth has never been exceeded in the conducted tests and it is therefore unclear what effect limited bandwidth has on the simulation performance.

Other tests have been conducted to measure the scalability of the co-simulation environment, see also [15]. An extendable pendulum model has been created that connects shafts with a ball bearing. The simplest meta-model contains three external models, i.e., two shafts and a bearing, one shaft that is connected with one end to the inertial system (ground) and with the other end to the inner ring of the bearing (using an external interface). The other shaft is connected to the bearing outer ring. The model has a total of two TLM connections, one between each shaft and the bearing. More bearings and shafts are gradually added to the pendulum to investigate the scalability of the co-simulation, see also Fig. 10. Different meta-models with up to five bearings connecting six shafts have been simulated, the result is shown in Table 2. One can see that an increasing amount of external models (and TLM connections) does not have a large impact on the total simulation time. Scalability of the meta-model simulator is considered to be good for this case. However, a more detailed performance analysis of the time spent in the different parts of the meta-model simulation, i.e., the solvers of the external models, the communication in the external interfaces, and the simulation manager, is planned for the future.

8. Conclusion

A general approach for mechanical system co-simulations has been presented. Several tool specific simulation models can be integrated and connected by means of a meta-model. The meta-model defines the physical interconnections of different external models, where an external model is a model defined in a specific language together with a modelling and simulation tool that can perform a simulation of it.

A modelling and simulation environment for meta-models has been created. Its main features are decoupled, lightweight, simulation manager and capabilities for storing simulation results. External model execution is automated and standardised, allowing for many different simulation tools to participate in the meta-model simulation. The simulation manager is kept small to avoid communication overhead in the coupled simulation. The approach is based on a general external interface definition that can be implemented for many different simulation tools and uses the TLM (Transmission Line Modelling) method for numerically stable co-simulation. It has successfully been implemented and tested for several simulation tools.

Acknowledgements

This work has been supported by SKF, the Swedish Foundation for Strategic Research (SSF/ProViking), the Excellence Center in Computer Science and Systems Engineering in Linköping (ECSEL), and the Knowledge Foundation (KK-stiftelsen).

References


