A. Siemers, I. Nakhimovski, D. Fritzson
Linköping University, Sweden

Meta-modelling of Mechanical Systems with Transmission Line Joints in Modelica

pp. 177-182

Meta-modelling of Mechanical Systems with Transmission Line Joints in Modelica

Alexander Siemers   Iakov Nakhimovski   Dag Fritzson
Linköping University, Linköping, Sweden
SKF, Göteborg, Sweden

Abstract

A framework for meta-modelling with *Transmission Line* (TLM) joints is presented. The framework is intended to support transient simulations of mechanical systems using co-simulation of different tools. The expressive power of the Modelica language 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 to an XML document which is accepted as input by the co-simulation engine. The framework prototype for SKF's BEAST and MSC.ADAMS is presented here. It is designed to be general, so that support for other simulation tools can be easily added. The main focus is on modelling of co-simulation Meta-Models taking advantage of Modelicas graphical and object-oriented modelling capabilities.

Keywords: simulation; co-simulation; meta-modelling; multibody; TLM; XML

1 Motivation

In the area of modelling and simulation of mechanical systems one can identify many different classes of models and corresponding tools. The specialization leads to different focus for different tools. One might say that every tool is optimized for a certain kind of problems. In terms of meta-modelling every tool can be seen as a black-box handling a particular *component*. A component is a model defined in some specific language together with some modelling and simulation tool that can perform a transient simulation of it. The examples of such components are equation-based multi-physics Modelica models, general multibody models in MSC.ADAMS, models with detailed contact definitions in SKF's BEAST, flexible components as modelled in FE tools, etc. .

In reality the different components are dependent on each other. Two components that are in physical interaction form boundary conditions for each other and some interface can often be defined.

Unfortunately it is often the case that the different classes of tools are used independently. Every class of tools is using approximations of the components it has interface with, that is, simplified models of the boundary conditions. Several time consuming iterations are often necessary to make the components converge to similar values on on the common interface. The limitations on the modelling accuracy are thus fundamental.

The need to bring different components into a complete more tightly coupled simulation is therefore justified. This allows higher accuracy and preserve the investments in the components.

Different co-simulation systems have appeared on the market during the last years. Most of them are focused on co-simulation of control systems and corresponding mechanical component. The coupled simulations this paper is focusing on are different. All components in our framework are mechanical and they have forces and motion in the interfaces. What is more important from numerical point of view, the sub-models are likely to use different differential equation solvers with variable time step. Numerical stability, which is not an issue for discrete time simulations, becomes an important consideration.

One method that was earlier used to enable closer interaction between such sub-models in a coupled simulation is *transmission lines modelling* (TLM). The TLM uses physically motivated time delays to separate the components in time and enable efficient co-simulation. The technique has proven to be stable and was implemented for coupling of hydraulical and mechanical sub-systems [1], [2].

However, no attempt to design a general coupled simulations framework was done. In this paper a general approach to meta-modelling of mechanical systems using TLM is presented. Modelica language is used to make such models easy to manage and the frame-

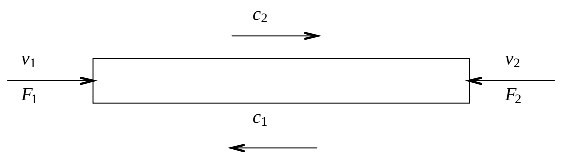


Figure 1: Delay line with the passing wave variables c_1 and c_2 and velocity variables v_1 and v_2.

work is designed to enable simple extension with new simulation tools.

2 Transmission Line Modelling

TLM method, also called Bilateral Delay Line Method, exploits the fact that all physical interactions in nature have finite propagation speed. The properties of the delay lines were studied in [7]. The method is briefly described below.

A basic one-dimensional transmission line is shown in Figure 1. For the mechanical case the line is basically a long spring with force waves c_1 and c_2 going between it ends. The input disturbances are velocities v_1 and v_2 and the reaction forces from the transmission line F_1 and F_2.

Note that the spring in our implementation is assumed to be iso-elastic. That is no cross-term waves are generated when working in 2D and 3D. See [2] for further discussions.

If the line delay is set to T and its impedance to Zc then the govering equations are:

$$\begin{aligned} c_1(t) &= F_2(t-T) + Zc\, v_2(t-T) \\ c_2(t) &= F_1(t-T) + Zc\, v_1(t-T) \\ \\ F_1(t) &= Zc\, v_1(t) + c_1(t) \\ F_2(t) &= Zc\, v_2(t) + c_2(t) \end{aligned} \tag{1}$$

The equations show that the two simulation systems are decoupled with the delay time T. Simulation framework can utilize this decoupling to enable efficient communications during co-simulation.

The transmission line introduces a parasitic mass $m_{tlm} = Zc\, T$ and stiffness $k_{tlm} = Zc/T$. Since it is often necessary to have a relatively large delay time (to enable larger communication intervals) while keeping the stiffness value, the user needs to be aware of the large parasitic mass.

3 Simulation Framework

The design goals for the simulation part of the framework were portability, simplicity to incorporate new simulation tools, computational efficiency. The design goals were realized by defining following concepts and interfaces:

TLM interface. A named point on a mechanical object where position and velocity can be evaluated and reaction force applied.

TLM manager. The central simulation engine. It is a stand alone program that reads in a XML definition of the coupled simulation. It then starts *Simulation components* and provides the communication bridge between the running simulations. That is the components only communicate with the TLM manager which acts as a broker marshalling information between the components as required by TLM theory. TLM manager sees every simulation component as a black box having one or several TLM interfaces. The information is then forwarded between TLM interfaces belonging different components.

TLM plug-in. A small $C++$ library having a single abstract class representing TLM interface for a specific simulation tool. The TLM plug-in can be seen by a simulation component as an external force that depends on position, velocity and time. The implementation of the plug-in handles the necessary communications with TLM manager.

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

4 Modelica as Meta-Model Language

Simulations of complete systems where components are modeled and simulated in different simulation packages are called co-simulations. The model of a co-simulation including all system components and its inter connections we call a meta-model.

The extended markup language (XML) has its strength in textual data representation and conversion. It is often the language of choice for communicating information between different tools. Those were the reasons behind the decision to use it as the input to the simulation engine.

Readability and edit-ability, on the other hand, are not the strengths of XML. Design of co-simulation meta-models requires thus a more powerful modelling language or graphical modelling environment. The following requirements were defined for meta-model definitions:

- Meta-Models should be based on a standard language.
- A graphical model editor should be available for ease of use.

Modelica with its object oriented modelling capabilities and its standardized graphical notations is thus perfectly suited. The fact that the Modelica standard defines graphical notations results in the availability of graphical model editors, i.e., MathModelica [5] and Dymola [4]. These editors typically allow easy connection modelling and user interface driven class design.
It should be mentioned that only Modelicas modelling capabilities are of interest here. Meaning that there is no need for Modelica based simulations. The use of Modelica as meta-modelling language might as well simplify the integration of Modelica simulations into meta-model based co-simulations. This, however, is not within the scope of this work.

4.1 Meta-Model Class Library

A meta-model Modelica package for component and TLM connection modelling, using Modelicas object oriented features, has been designed.
Three packages plus a base model class were defined:

The Components package contains classes for the different simulation components. These are currently BEAST and MSC.Adams components.

The Connections package contains the TLM connection or joint. Different TLM specific parameters can be specified for each connection.

The Interfaces package contains the corresponding TLM interface. Each TLM component contains at least one TLM interface.

The BaseMetaModel class is the base class for each Meta-Model. It contains Meta-Model specific parameters.

Different TLM components are defined in the components package which are inherited from the simulation tool specific components, see also Figure 2.

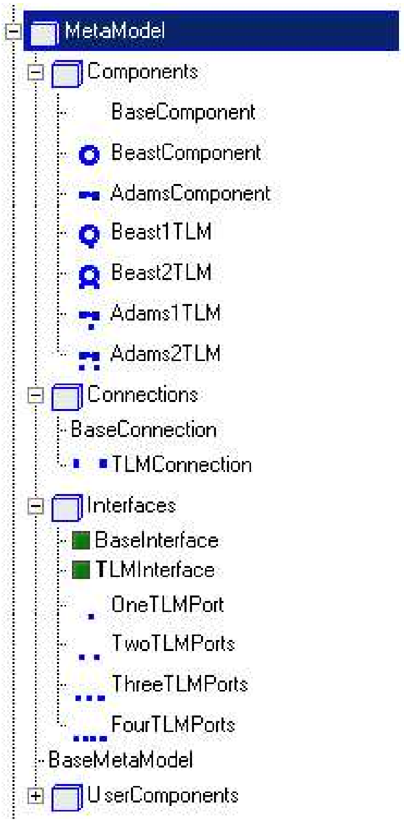


Figure 2: The basic TLM Meta-Model class library.

They add a certain number of TLM interfaces to each component. TLM connections define data exchange and synchronization between these components during co-simulation. Connections are created between two TLM interfaces of two TLM components. TLM interfaces are therefore defined as `connectors`.
Several base classes define common model parameters needed by the TLM manager or for correct XML translation. Specialized child classes modify these parameters to their needs. BEAST Components for example modify the start-method as follows:

```
model BaseComponent
  parameter String Description;
  parameter String SimulationFiles;
  parameter String StartMethod;
     .
     .
end BaseComponent;

model BeastComponent
  extends BaseComponent
(StartMethod="beast --serial");
     .
     .
end BeastComponent;
```

Both component and interface classes contain a type specifier which is `TLM` for TLM components and TLM interfaces. This allows for additional type checking during model translation and guarantees that TLM interfaces are connected with TLM connections. But is also useful for future extensions with new connection types.

4.2 Component Modelling

Component modelling is divided into two steps:

- Component modelling in the specialized environment. Each component of the multi-scale simulation is modeled in its specific environment. Users define the TLM interfaces to the model.
- Component modelling in the multi-scale environment. The component needs to be integrated into the multi-scale environment. Startup methods, interfaces, and communication parameters must be specified.

Co-simulation components are modelled in the modelling environment of the specific simulation tool. To participate in a TLM co-simulation each simulation program needs to integrate an TLM plug-in and a way to model TLM interfaces. In MSC.ADAMS for example external forces are connected to a TLM interface, and BEAST defines so called TLM-ties. The TLM interfaces are thus part of the simulation model expressed in the modelling language of the specific program.
Simulation components are integrated into the meta-model by selecting a matching component from the Modelica Meta-model library. Component type and number of TLM interfaces have to match.

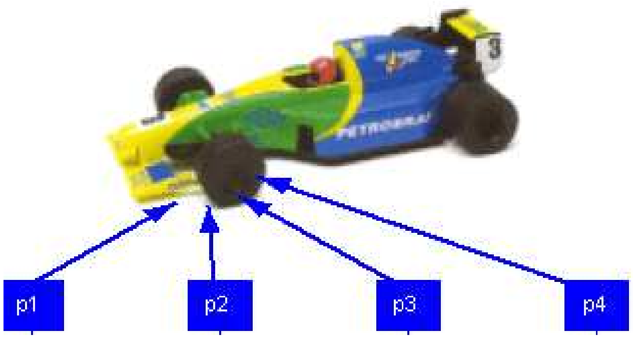


Figure 3: A MSC.ADAMS car simulation component in the Modelica environment with four TLM interfaces at the front tire.

Alternatively can the base components, i.e., *BeastComponent* and *AdamsComponent*, be extended (inherited) and a certain number of TLM interfaces be added. This allows for other extensions as well, e.g., selecting appropriate component icons for more intuitive modelling, see Figure 3. New components should be added to the *UserComponents* package in the Meta-Model library. This is needed for the XML translator to work properly.

4.3 Meta Modelling

Meta-Models are created using a graphical Modelica editor, see Figure 4, where components are dragged into the model. Every Meta-Model must extend the `BaseMetaModel` class that contains Meta-Model and co-simulation specific parameters. TLM components and connections are added to the model and connections are drawn between the TLM interfaces.

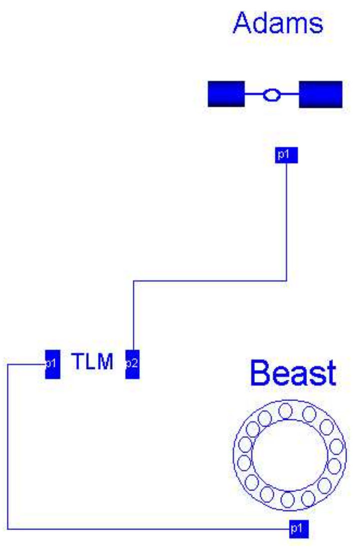
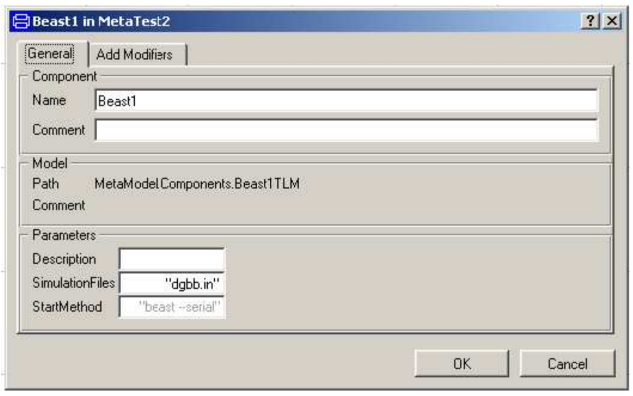


Figure 4: A simple BEAST-MSC.ADAMS Meta-Model.

Several parameters need to be specified for the different parts of the model. They are needed by the TLM manager for correct simulation execution. BEAST and MSC.ADAMS components, for example, need a simulation file to be specified, see Figure 5, and TLM connections require correct TLM parameters.
The meta-model description is kept general and works with any simulation tool that supports TLM connections. New components can be created by extending the BaseComponent class or any of the predefined component classes. Only the start-method for the simulation tool needs to be specified for new components. Predefined components can be extended if more TLM interfaces are required. The number of required TLM interfaces is application and simulation-model dependent.


Beast1 in MetaTest2

General | Add Modifiers

Component
Name: Beast1
Comment:

Model
Path: MetaModelComponents.Beast1TLM
Comment:

Parameters
Description:
SimulationFiles: "dgbb.in"
StartMethod: "beast --serial"

OK | Cancel

Figure 5: The BEAST component parameter dialog.

4.4 Meta-Model Translation

The meta-model is translated into XML code to run in the co-simulation framework. A Modelica to XML translator has been designed for this purpose. The translator makes use of ModelicaXML [3] plus some co-simulation specific translations. The translation is done in two steps:

1. Translation from Modelica to ModelicaXML
2. Translation from ModelicaXML to the Meta-Model XML representation

To simplify parsing of the Modelica Meta-Model it is first translated into a Modelica-XML representation using the ModelicaXML [3] translator. This representation can be parsed and translated with a standard XML-parser. The libXML2 [6] standard library has been used to convert the ModelicaXML Meta-Model into the XML representation required by the TLM manager.

4.5 Meta-Model Example

An typical example of a BEAST-MSC.ADAMS Meta-Model is shown in Figure 6. A front wheel bearing hub-unit is connected to the race-car with four flanges each of which is modelled as a TLM connection. The components have to be prepared in BEAST and MSC.ADAMS to contain the TLM interfaces. Afterwards they are integrated into the meta-model environment by creating component classes with appropriate icons and TLM interfaces in the Modelica package. Each Meta-Model needs to extend the *BaseMetaModel* Modelica class to inherit the global co-simulation parameters. TLM connections are added between the TLM interfaces according to the hub-unit flanges. The complete Modelica model looks like this:

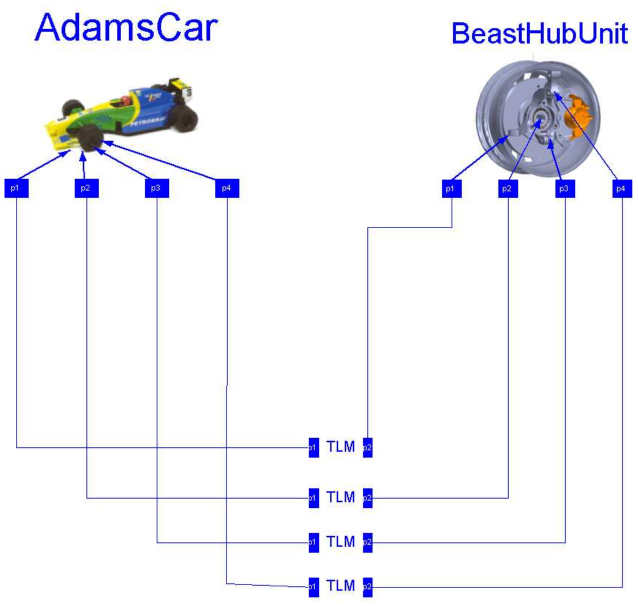


Figure 6: Modelica Meta-Model detailed BEAST hub unit integrated into a MSC.ADAMS racing-car model.

```
model BeastHubInAdamsCar
  extends MetaModel.BaseMetaModel;
  MetaModel.UserComponents.BeastCarCorner
    BeastHubUnit(
     Description=
        "A complete Beast hub-unit",
     SimulationFiles="CarCorner.in",
     StartMethod="start-beast");
  MetaModel.UserComponents.AdamsCarModel
    AdamsCar(
     Description="A MSC.ADAMS car model",
     SimulationFiles="racing_car.cmd");
  MetaModel.Connections.TLMConnection TLM1;
  MetaModel.Connections.TLMConnection TLM2;
  MetaModel.Connections.TLMConnection TLM3;
  MetaModel.Connections.TLMConnection TLM4;
equation
  connect(AdamsCar.p1,
  TLMConnection1.p1);
  connect(AdamsCar.p2,
          TLMConnection2.p1);
  connect(AdamsCar.p3,
          TLMConnection3.p1);
  connect(TLMConnection1.p2,
          BeastHubUnit.p1);
  connect(TLMConnection2.p2,
          BeastHubUnit.p2);
  connect(TLMConnection3.p2,
          BeastHubUnit.p3);
  connect(TLMConnection4.p2,
          BeastHubUnit.p4);
  connect(AdamsCar.p4,
          TLMConnection4.p1);
end BeastHubInAdamsCar;
```

Finally the model is converted into the XML representation required by the TLM manager by first converting it into ModelicaXML and then into the XML Meta-Model representation. The XML model looks like this:

```
<?xml version="1.0"?>
<Model name="BeastHubInAdamsCar"
       StartTime="0"
       EndTime="1"
       TLMDelay="0.001">
  <SubModels>
    <SubModel Name="BeastHubUnit"
       Description=
        "A complete Beast hub-unit"
       SimulationFiles="CarCorner.in"
       StartMethod="start-beast">
      <InterfacePoint Name="p1"
         iType="TLM"/>
      <InterfacePoint Name="p2"
         iType="TLM"/>
      <InterfacePoint Name="p3"
         iType="TLM"/>
      <InterfacePoint Name="p4"
         iType="TLM"/>
    </SubModel>
    <SubModel Name="AdamsCar"
       Description=
          "A MSC.ADAMS car model"
       SimulationFiles="racing_car.cmd"
       StartMethod="start-adams">
      <InterfacePoint Name="p1"
         iType="TLM"/>
      <InterfacePoint Name="p2"
         iType="TLM"/>
      <InterfacePoint Name="p3"
         iType="TLM"/>
      <InterfacePoint Name="p4"
         iType="TLM"/>
    </SubModel>
  </SubModels>

  <Connections>
    <Connection From="AdamsCar.p1"
      To="BeastHubUnit.p1"
      iType="TLM" alpha="0" Zf="0"/>
    <Connection From="AdamsCar.p2"
      To="BeastHubUnit.p2"
      iType="TLM" alpha="0" Zf="0"/>
    <Connection From="AdamsCar.p3"
      To="BeastHubUnit.p3"
      iType="TLM" alpha="0" Zf="0"/>
    <Connection From="BeastHubUnit.p4"
      To="AdamsCar.p4"
      iType="TLM" alpha="0" Zf="0"/>
  </Connections>
</Model>
```

5 Conclusion

A framework for meta-modelling and simulation of mechanical systems using transmission lines for coupling components was presented. The main features of the framework are:

- General object-oriented meta-modelling utilizing the strengths of Modelica
- Stability by applying minimalist approach to the program design resulting in small portable code
- Extensibility of the framework thanks to the portable and easy to incorporate software plug-in.

The framework currently targets SKF's BEAST simulation tool and MSC.ADAMS.

References

[1] Krus P., Jansson A. Distributed Simulation of Hydromechanical Systems 'Third Bath International Fluid Power Workshop', Bath, UK 1990

[2] Krus P. Modelling of Mechanical Systems Using Rigid Bodies and Transmission Line Joints. Transactions of ASME, Journal of Dynamic Systems Measurement and Control. Dec 1999

[3] Pop A., Fritzson P. ModelicaXML:A Modelica XML Representation with Applications, Modelica 2003 Conference

[4] Dymola, *http://www.dymola.com*, Dynasim AB

[5] MathModelica, *http://www.mathcore.com/products/mathmodelica* Mathcore AB

[6] The XML C parser and toolkit of Gnome, *http://www.xmlsoft.org/*

[7] Larsson, J. Interoperability in Modelling and Simulation, PhD thesis, Link¨oping University, Link¨oping, Sweden, 2003