Modelica-Association-Project "System Structure and Parameterization" – Early Insights

Jochen Köhler[1] Hans-Martin Heinkel[2] Pierre Mai[3] Jürgen Krasser[4] Markus Deppe[5] Mikio Nagasawa[6]

[1]ZF Friedrichshafen AG, Germany, jochen.koehler@zf.com
[2]Robert Bosch GmbH, Germany, Hans-Martin.Heinkel@de.bosch.com
[3]PMSF IT Consulting, Germany, pmai@pmsf.eu
[4]AVL List GmbH, Austria, juergen.krasser@avl.com
[5]dSPACE, Germany, MDeppe@dspace.de
[6]CYBERNET SYSTEMS Co., Ltd., Japan, mikio-n@cybernet.co.jp

Abstract

Starting with the motivation to invent the new standard SSP ("System Structure and Parameterization") within the Modelica Association and the need to have one more standard beyond the mature Modelica language and the already well established Functional Mockup Interface (FMI) proposed in Modelica Association (Blochwitz *et al*, 2011), the main use-cases are presented were SSP can help. As SSP relies on XML, the schemas and in consequence the main features for defining system structures and parameterization of models are described. The need to be able to transport complex networks of FMUs between different simulation platforms like MIL, SIL and HIL is emphasized as a motivator for SSP.

A variety of prototypes are shown that support the early version of SSP. This gives a good impression how the standard can be used for quite different tasks and proofs, that system structures can be exchanged between them seamlessly.

Finally the next steps for the ongoing development of SSP are outlined.

Keywords: FMI, System Structure, Parameterization, Collaboration, Standardization

1 Introduction

It's still a very big challenge for different kinds of industry areas to build up simulation models for behavioral simulation of complex systems consisting of multiple domains. In the engineering process we are used to separate a system into its components and do all the necessary simulations for one component in a tool that fits best to the specific problem. Good results can be achieved in this way if the dynamic behavior of one component has no large impact on the other parts of the system. But as the systems to be developed become more complex and the interaction between all components becomes more important or is even essential for the product value the simulation of the connected parts is inevitable. Figure 1 shows a typical example from the automotive industry, where component models have to be combined for overall simulation in different environments.

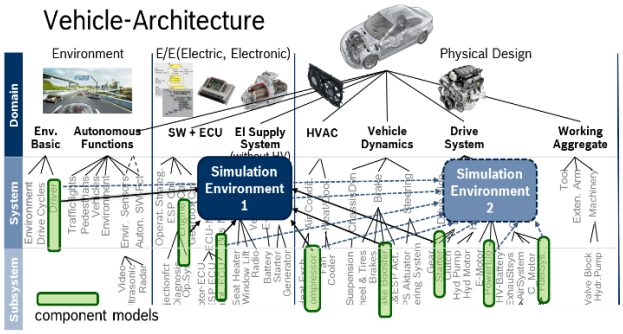

Figure 1. System simulation in automotive industry

The attempt to model all physical domains within one tool could be a potential solution, e.g. modeling languages like Modelica can handle this quite well.

One large benefit here is the possibility that during the translation process of the complete system a lot of mathematical simplification mechanisms can be used to optimize the mathematical problem and make it easier to solve the DAE during simulation with one single solver. However due to the complexity of the language and the fact, that other simulation tools are quite more established in certain domains it is very hard to enforce this approach in a company, or for collaborative development across companies.

The second best approach came with FMI. The standardized Functional-Mockup-Interface gives the possibility to export an FMU (Functional mockup unit) of a component from the authoring tool that was used to build it and integrate it in another environment to simulate it. Of course in this integration environment other component FMUs can be integrated as well to connect them all into a complete virtual system. But once again this can be done only in a proprietary way

for this single integration tool and the built-up system structure cannot be transferred to another environment. However, this is a common use case when the modeled system has to be used in different targets like MIL, SIL or HIL.

Another issue when simulating complex systems is parameterization. FMUs can be parameterized in an isolated way but there is no convenient way to handle these parameters e.g. exchanging complete parameter sets or handling any associated intellectual property concerns. To parameterize complete systems a "global" instance has to exist to handle all the parameters that are not part of a component or have to be used in several components at the same time. These features are quite relevant when models of components are interchanged between different departments in one organization or - even more important – between different companies.

These should not be considered as disadvantages of the FMI approach because FMI does not have these issues in its scope.

These thoughts were presented first at the Modelica Design Meeting in 2014 in Lund by BMW, Bosch and ZF and there was a commitment to instantiate a new Modelica Association project called "System Structure and Parameterization" to develop mechanisms and standards to enhance the existent FMI standard. It is important to emphasize that this new standard is developed in close cooperation with the FMI project group to secure a perfect fit of both standards.

In the last months there was quite good progress starting with the definition of different use-cases to describe various scenarios handling system structures and parameters. Derived from that a number of XML schemas have been developed to describe both system structures and the parameterization of complete systems in a standardized way that can be used independently of specific tools.

First evaluation of this could be shown at the Modelica Conference 2015 in France where three different tools were presented that were able to read in the same XML representation of a simulation model and handle it in their specific ways.

The next sections give detailed insights into the use cases, the actual status of the XML schemas and a short presentation of the already existing tool prototypes that make use of this upcoming standard.

2 Use Cases

In the following chapter the basic use cases and the principal solutions by the SSP-project are described

2.1 Parameter Exchange

In future, updates and effective variant handling for models will be done predominantly by parameter sets. To do this effectively in a heterogeneous environment, we need a tool independent standard. Figure 2 and Figure 3 show the use cases and possible solution for a single model and for a structure of models

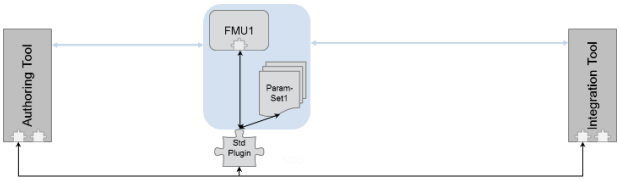

Figure 2. Exchange of one FMU/model with multiple different parameter sets

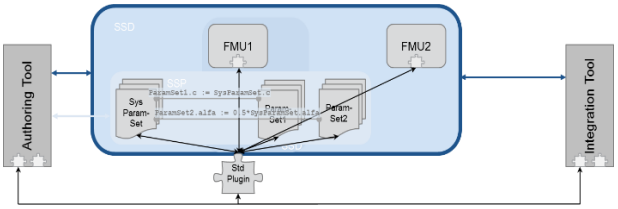

Figure 3. Describing parameter sets for system architecture

2.2 Model Structure

As shown in Figure 1 the multiple use of (sub-) structures of models in heterogeneous environments get more important. For the seamless and tool independent usage of networks of components, we need a standardized format for the connection structure, which also support basic mathematical manipulation of signals (for manual unit conversion or mapping of discrete signals). Figure 4 shows the approach of the SSP project.

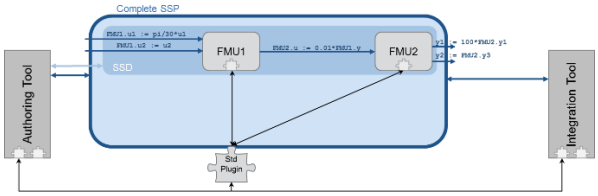

Figure 4. System architectures with signal modifications

2.3 Model Structure and Parametrization

Use case 2.3 is the combination of use case 2.1 and 2.2 (Figure 5). The structure and the according parameter sets have to be handled in a tool independent standard.

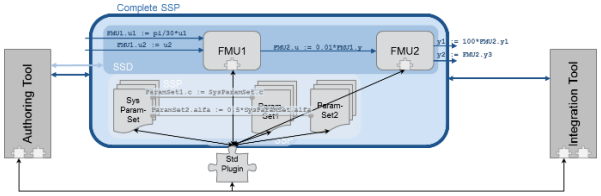

Figure 5. System architectures with signal adoption layer and parameter sets

3 XML Schemas

The file formats defined for the MAP-SSP project are intended to provide a minimal interchange format between different tools, not as a replacement for tool-specific formats, and are focused on the exchange of information on systems needed for their execution or integration into other systems. The interchange of architectural information between architecture tools (e.g. SysML-based tools, for which XMI already provides an interchange format) is out of focus of the current efforts,

The formats try to duplicate as little information as possible from any referenced component formats, like FMUs, and try to be agnostic as to the detailed semantics of the connections being described, submitting to the semantics definitions of the relevant standards for e.g. FMUs for actual connection semantics. In this way the format should be useful for many different purposes and should potentially be compatible with currently envisaged FMI standard developments, like e.g. structured ports.

By defining both a basic system structure file format (SSD) and a format for packaging the SSD and its related resources, including referenced SSDs/SSPs and FMUs into an easily transportable archive (SSP), the proposal tries to offer flexibility in the way system structure is being exchanged in different contexts, e.g. within companies using PLM systems or between companies in a customer/supplier context.

Currently XML schemas have been defined for the description of the system structure itself (System Structure Definition – SSD, file extension .ssd), of parameter sets (System Structure Parameter Values – SSV) and their mapping to system/component parameters (System Structure Parameter Mapping – SSM). Additionally the System Structure Package format (SSP, file extension .ssp) is defined, which constitutes a ZIP-archive that packages together a set of system structure definitions and any referenced parameter sets, mappings, components and sub-systems into one easily handled and transferable unit.

A SSP must contain at least one SSD file, but can contain multiple such files at top-level, which give the ability to package multiple variants of a system into one SSP, allowing the importing user/tool the selection of which variant to process. This enables the efficient exchange of systems/sub-systems with varying system topology, e.g. for vehicle models with different propulsion systems and architectures, while being able to reuse commonly shared resources like sub-systems, FMUs, or parameter sets.

The SSD file defines the structure of a system: Its external interface (if any), i.e. the system input, output and parameter connectors as exposed to the outside, and the internal structure, including instantiated components, like FMUs or referenced external systems, subsystems, as well as connections between components and between components and the external interface.

For each component any referenced inputs, outputs and parameters are specified as connectors as well. Connections between connectors that are physical quantities will perform unit conversions by default. Connections can also apply linear transformations (for continuous quantities) or mapping transformations (for discrete quantities) in order to adjust values between components as needed.

The system description also assigns parameter sets (SSV) to components or complete (sub-)systems, either with a natural 1:1 mapping or by specifying explicit parameter mappings in the SSM format. See Figure 6 for a simplified overview of the data model behind the XML schema and Figure 11 for a simple example file.

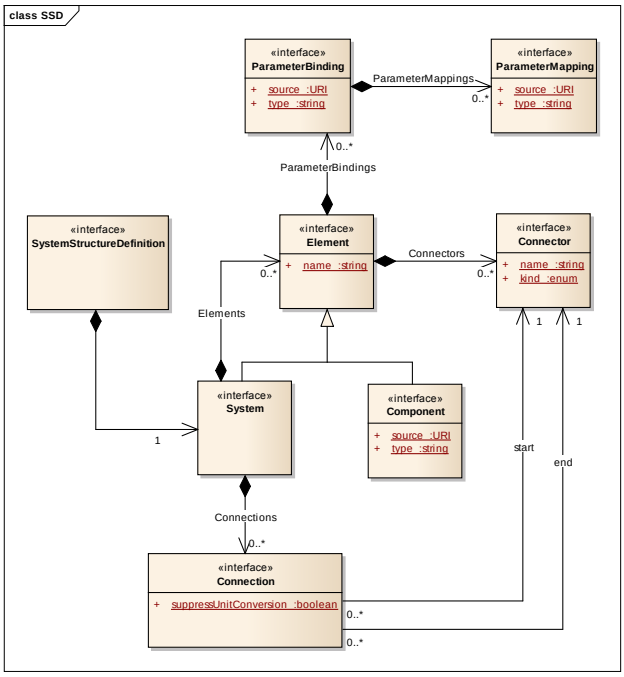

Figure 6. Simplified Class Diagram for SSD Schema.

In order to support exchange of system structure between tools that offer a graphic view of a system, optional geometric information for systems, components, connectors and connections is supported.

The SSV format defines a parameter set, consisting of a set of parameter definitions, including parameter values and related meta-information (like data type, physical unit), as necessary to aid in the exchange of parameter sets and their use in parametrizing systems and components. A core set of meta-information is likely to be included in the final standard with extension mechanisms to support the exchange of user-specific meta-information as needed. Parameter sets in the SSV format can be contained directly in the relevant parameter binding element of the SSD file or referenced as an external .ssv file.

The SSM format, as illustrated in Figure 7, defines a mapping between the parameters in a parameter set and

the parameters of a component or system (potentially including subsystems and components) by mapping the names of parameters between the two namespaces and optionally providing further transformations on parameter values, like mapping of enumerations or linear transformations of continuous values. Like the SSV format this mapping can be contained within an SSD file or referenced as an external .ssm file.

All formats offer easy extensibility for specific tool or user needs through optional tool- or usage-specific annotations on all modeling elements. This also allows the simple addition of layered standards on top of the current formats.

References between SSD files and related resources, like components, parameter sets or parameter mappings are implemented as (relative) URIs. This allows the integration of these formats into larger resource management systems like PLM systems, so that e.g. SSD and/or referenced parameter sets or component FMUs can be located via HTTPS or PLM-specific URI schemes, in addition to the default file-based access mechanisms.

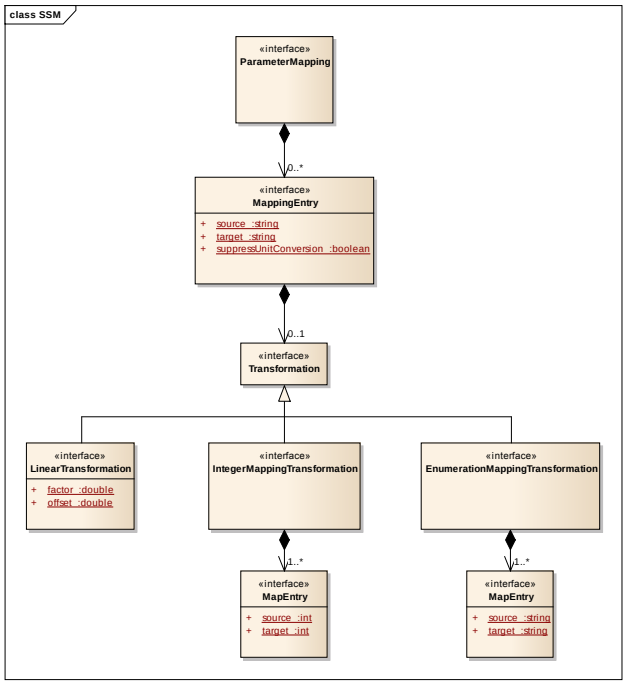

Figure 7. Simplified Class Diagram for SSM Schema.

Currently design discussions are on-going on the support of additional connection constructs, like signal dictionaries or bus-like connections, which aid in the maintenance of component and system interconnections with many, frequently changing signals, like those employed by bus communication mechanisms between controller models. It is expected that the initial release of the SSP standard will include such mechanisms.

While the work in the SSP project was initially focused on FMU-based systems, the SSD and SSV/SSM formats are intentionally also suitable for describing systems containing other component types, like models or controller code, if relevant definitions are implemented for these component types.

4 Hardware-in-the-Loop Simulation

In order to cope with the growing complexity of modern electronic control units (ECUs), Model-Based Design (MBD) is used throughout the embedded software development process. The result is an increasing number of models designed for various purposes. During the MBD process different methods are applied to test the software of an ECU. In early stages PC-based model-in-the-loop (MIL) and software-in-the-loop (SIL) simulations are commonly used to validate the software. Additionally hardware-in-the-loop (HIL) simulation based testing is applied as the tried-and-tested method for function, component, integration and network tests of an entire system. HIL simulation is an integral part of the development process of many OEMs and suppliers across different industries. Due to the inclusion of real hardware in the test setup, HIL test systems have special requirements that do not allow the same free choice of simulation methods as for MIL/SIL use cases. Usually specialized HIL simulation systems with optimized hardware and real-time operating systems (e.g., QNX, Linux-RT) are necessary. These systems have to meet real-time requirements and handle the system dynamics and timing of the ECU computation timing loops. Common HIL applications typically require hard real-time, fixed-step solvers with sampling times of 1ms or less.

FMUs for co-simulation are a good basis for the tool- and platform-independent exchange of simulation models in HIL environments. The lean co-simulation interface reduces possible compatibility issues in a tool chain that includes various FMI supporting tools. Moreover, it systematically separates the FMU functions from the tool functions. This separation enables efficient FMU internal implementations of e.g. tunable parameter support and internal multi-rate subsampling. These co-simulation FMUs can transport verified combinations of solver and model code. Additionally the communication point concept can separate internal solver steps from external communication steps. FMUs may include ANSI-C source code, which is important for platform-independent reuse, but often conflicts with modelers' and tool vendors' interest in protecting their IP. Exported FMUs therefore often only contain precompiled binaries and are consequently limited to specific pre-selected target platforms.

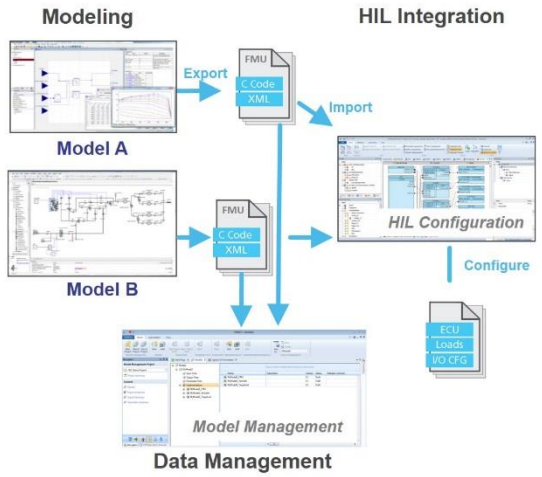

Figure 8. Integration of FMUs for HIL Testing.

Today single FMUs are imported into HIL configuration tools to integrate them with other FMUs, Simulink-based models, Virtual-ECUs or real ECUs (Figure 8). HIL simulation tests with real-time capable FMUs can rely on the full functionality of a tool chain including test automation and visualization. Additionally newer Model-Based Design Test and Data Management environments provide capabilities for managing model compositions, handling variants of models and systems-under-test and managing the parameter and signal interfaces of the different model systems. These functions are necessary in order to optimize the usage of models and associated data assets throughout the lifecycle of development and validation. Especially for managing complex simulation scenarios resulting from the use of models from various modeling environments such capability is crucial.

Environment models are often developed and specially designed for certain use cases. However, there is an increasing desire to reuse these models to provide proven, consistent solutions for the validation of controller models in different projects and development stages (e.g., for virtual validation and HIL simulations). A reuse of models increases productivity and saves time by eliminating the duplication of design efforts. The environment models that are exchanged – e.g., based on FMI – need to be suitable for all the intended MIL/SIL/HIL simulation use cases. Modular design of models and particularly clear identification of interfaces pertaining to real or simulated components, are necessary to allow an exchange of simulated components by a real hardware component at various points in the development and testing processes. In co-simulation scenarios, a model structure should be chosen that separates the overall model into weakly coupled model parts that can be computed concurrently and are insensitive to input delays due to co-simulation effects.

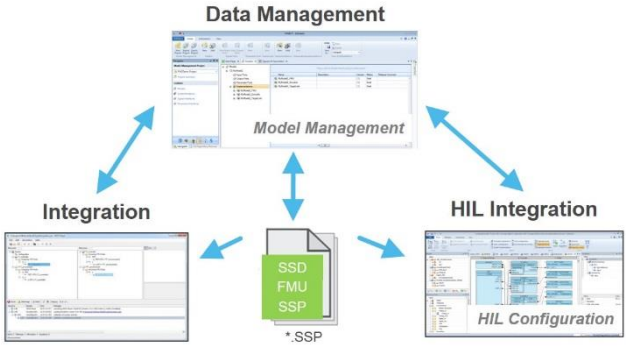

Figure 9. Potential ways to exchange the System Structure Description.

Once a reasonable model structure is designed there is no standardized way so far to exchange it among tools from different vendors especially if no overall integration model exists. The SSP approach allows to share a standardized system structure description between data management, integration and configuration tools for SIL, MIL and HIL scenarios (Figure 9). Hence, the SSP interchange format helps to improve the consistent simulation and interchange of complex models in the MBD process.

5 Prototypes

The specifications of SSD are investigated with some prototype tools assuming various co-simulation environments.

5.1 Integration Tool

Model.CONNECT™, a product of AVL List GmbH, is a tool to set up and execute system simulation models which are composed of subsystem and component models from multiple model authoring environments. Models can be integrated based on standardized interfaces (FMI) as well as based on specific interfaces to a wide range of well-known simulation software.

In order to validate the SSP specification, we implemented a prototype plug-in for the tool that supports the export and import of system configurations.

During the prototype development we particularly explored the capabilities of SSP to pack variants of system configurations into one archive. We found that the very basic variant support in SSP could be mapped to/from the sophisticated variant management capabilities in the tool, which is designed to describe both different configurations of the system under investigation as well as different testing scenarios and testing environments. It is important to mention, that the variant handling in SSP is deliberately and by design not expressive enough to support loss-less export-import roundtrips (e.g. Model.CONNECT™→ SSP → Model.CONNECT™) with respect to variant management. We plan to address this in future by

enriching the export/import plug-in based on tool-specific annotations in the SSP.

This prototype SSP export/import plug-in was also used to test the specifications of SSP with regards to the graphical representation (as 2D block model) of system configurations. Also here we did not encounter any major issues mapping between SSP and the tool-specific geometry handling. This is again a result of SSP design principle to keep the specification as simple as possible: SSP allows to transfer component and connector positions as well as connection way-points. Thus, SSP allows to re-construct the main aspects of a layout, but it makes no attempt towards a pixel-by-pixel identical rendering of systems in the exporting and importing systems.

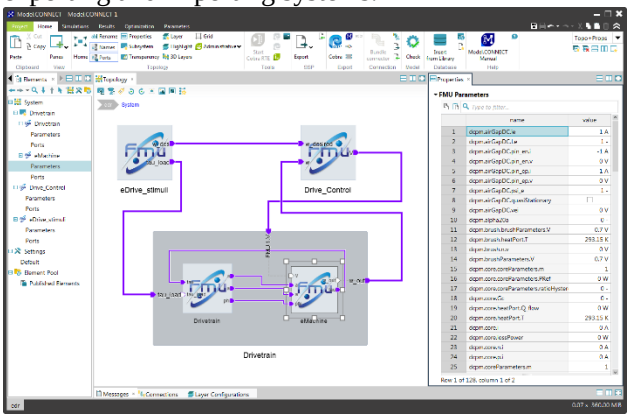

Figure 10 eDrive example in Model.CONNECT™. The subsystem "Drivetrain" is displayed in transparent mode to see its internal structure.

Future work on the plug-in will be focusing on the support of parameter values and mappings. Workflow-wise, we will explore using SSP to transfer system information from SysML-based MBSE tools to Model.CONNECT™.

5.2 Co-Simulation Browser

FMI has become a common model exchange and co-simulation standard. However the master-level runtime verification and validation of the virtual system made of many slave models are still difficult for FMI users. The integration of multi-domain expertise is required to analyze the multi-FMU complexity and the large scale virtual system simulation results. In order to facilitate the system-level FMU user collaboration, a light weight FMI/SSP Co-Simulation browser is prototyped. This co-simulation browser includes the simulation player and the parser of SSP defined System Structure XML such as in Figure 11 .

```
<?xml version="1.0" encoding="UTF-8"?>
<ssd:SystemStructureDescription xmlns:ssd="http://www.pmsf.net/xsd/SystemStructureDescriptionDraft"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="Draft20150721" xsi:schemaLocation="http://www.pmsf.net/xsd/Syster
  <ssd:System name="eDriveSampleSystem" description="eDrive sample with geometry">
    <ssd:SystemGeometry x="0.0" y="0.0" width="0.0" height="0.0" canvasWidth="2.78" canvasHeight="2.41"/>
    <ssd:Connectors> <ssd:Connector name="ExternalOut" causality="output"/> </ssd:Connectors>
    <ssd:Elements>
      <ssd:System name="Drivetrain" description="">
        <ssd:SystemGeometry x="0.398" y="1.18" width="2.35" height="1.25" canvasWidth="2.07" canvasHeight="0.86"/>
        <ssd:Connectors>
          <ssd:Connector name="FMU 1.V" causality="input"> <ssd:ConnectorGeometry x="0.598" y="0.0"/> </ssd:Connector>
          <ssd:Connector name="tau_load" causality="input"> <ssd:ConnectorGeometry x="0.0" y="0.554"/> </ssd:Connector>
          <ssd:Connector name="w_out" causality="output"> <ssd:ConnectorGeometry x="1.0" y="0.431"/> </ssd:Connector>
        </ssd:Connectors>
        <ssd:Elements>
          <ssd:Component name="Drivetrain" type="application/x-fmu-sharedlibrary" source="resource/eDrive_Drivetrain.fmu">
            <ssd:ComponentGeometry x="0.0" y="0.0" width="0.658" height="0.658"/>
            <ssd:Connectors>
              <ssd:Connector name="tau" causality="input"> <ssd:ConnectorGeometry x="0.0" y="0.375"/> </ssd:Connector>
              <ssd:Connector name="tau_load" causality="input"> <ssd:ConnectorGeometry x="0.0" y="0.625"/> </ssd:Connector>
              <ssd:Connector name="a" causality="output"> <ssd:ConnectorGeometry x="1.0" y="0.25"/> </ssd:Connector>
              <ssd:Connector name="w" causality="output"> <ssd:ConnectorGeometry x="1.0" y="0.5"/> </ssd:Connector>
              <ssd:Connector name="phi" causality="output"> <ssd:ConnectorGeometry x="1.0" y="0.75"/> </ssd:Connector>
            </ssd:Connectors>
          </ssd:Component>
          <ssd:Component name="eMachine" type="application/x-fmu-sharedlibrary" source="resource/eDrive_eMachine.fmu">
            <ssd:ComponentGeometry x="1.45" y="0.0372" width="0.62" height="0.82"/>
            <ssd:Connectors>
              <ssd:Connector name="V" causality="input"> <ssd:ConnectorGeometry x="0.0" y="0.2"/> </ssd:Connector>
              <ssd:Connector name="a" causality="input"> <ssd:ConnectorGeometry x="0.0" y="0.4"/> </ssd:Connector>
              <ssd:Connector name="w" causality="input"> <ssd:ConnectorGeometry x="0.0" y="0.6"/> </ssd:Connector>
              <ssd:Connector name="phi" causality="input"> <ssd:ConnectorGeometry x="0.0" y="0.8"/> </ssd:Connector>
              <ssd:Connector name="w_out" causality="output"> <ssd:ConnectorGeometry x="1.0" y="0.3"/> </ssd:Connector>
              <ssd:Connector name="tau" causality="output"> <ssd:ConnectorGeometry x="1.0" y="0.7"/> </ssd:Connector>
            </ssd:Connectors>
          </ssd:Component>
        </ssd:Elements>
        <ssd:Connections>
          <ssd:Connection startElement="Drivetrain" startConnector="a" endElement="eMachine" endConnector="a">
            <ssd:ConnectionGeometry pointsX="1.05 1.05" pointsY="0.10 0.28"/> </ssd:Connection>
          <ssd:Connection startElement="Drivetrain" startConnector="w" endElement="eMachine" endConnector="w">
            <ssd:ConnectionGeometry pointsX="1.24 1.24" pointsY="0.33 0.47"/> </ssd:Connection>
          <ssd:Connection startElement="Drivetrain" startConnector="phi" endElement="eMachine" endConnector="phi">
            <ssd:ConnectionGeometry pointsX="1.25 1.25" pointsY="0.48 0.63"/> </ssd:Connection>
          <ssd:Connection startElement="Drivetrain" startConnector="tau" endElement="eMachine" endConnector="tau">
            <ssd:ConnectionGeometry pointsX="-0.17 -0.17 2.17 2.17" pointsY="0.25 -0.17 -0.17 0.35"/> </ssd:Connection>
          <ssd:Connection startConnector="FMU 1.V" endElement="eMachine" endConnector="V"/>
          <ssd:Connection startConnector="tau_load" endElement="Drivetrain" endConnector="tau_load"/>
          <ssd:Connection startElement="eMachine" startConnector="w_out" endConnector="w_out"/>
        </ssd:Connections>
      </ssd:System>
      <ssd:Component name="Drive_Control" type="application/x-fmu-sharedlibrary" source="resource/eDrive_Control.fmu">
        <ssd:ComponentGeometry x="2.01" y="0.0" width="0.8" height="0.8"/>
        <ssd:Connectors>
          <ssd:Connector name="w_desired" causality="input"> <ssd:ConnectorGeometry x="0.0" y="0.333"/> </ssd:Connector>
          <ssd:Connector name="w" causality="input"> <ssd:ConnectorGeometry x="0.0" y="0.667"/> </ssd:Connector>
          <ssd:Connector name="V" causality="output"> <ssd:ConnectorGeometry x="1.0" y="0.5"/> </ssd:Connector>
        </ssd:Connectors>
      </ssd:Component>
      <ssd:Component name="eDrive_stimuli" type="application/x-fmu-sharedlibrary" source="resource/eDrive_stimuli.fmu">
        <ssd:ComponentGeometry x="0.0" y="1.16e-08" width="0.8" height="0.8"/>
        <ssd:Connectors>
          <ssd:Connector name="w_des" causality="output"> <ssd:ConnectorGeometry x="1.0" y="0.333"/> </ssd:Connector>
          <ssd:Connector name="tau_load" causality="output"> <ssd:ConnectorGeometry x="1.0" y="0.667"/> </ssd:Connector>
        </ssd:Connectors>
      </ssd:Component>
    </ssd:Elements>
    <ssd:Connections>
      <ssd:Connection startElement="Drive_Control" startConnector="w_desired" endElement="eDrive_stimuli" endConnector="w_des">
        <ssd:ConnectionGeometry pointsX="" pointsY=""/>
      </ssd:Connection>
      <ssd:Connection startElement="Drivetrain" startConnector="tau_load" endElement="eDrive_stimuli" endConnector="tau_load">
        <ssd:ConnectionGeometry pointsX="0.068 0.068 0.70 0.70" pointsY="1.05 0.94 0.94 0.40"/>
      </ssd:Connection>
      <ssd:Connection startElement="Drive_Control" startConnector="V" endElement="Drivetrain" endConnector="FMU 1.V">
        <ssd:ConnectionGeometry pointsX="3.11 3.11 1.66" pointsY="0.30 0.83 0.83"/>
      </ssd:Connection>
      <ssd:Connection startElement="Drive_Control" startConnector="w" endElement="Drivetrain" endConnector="w_out">
        <ssd:ConnectionGeometry pointsX="1.80 1.80 3.12 3.12" pointsY="0.40 0.96 0.96 1.70"/>
      </ssd:Connection>
    </ssd:Connections>
  </ssd:System>
</ssd:SystemStructureDescription>
```

Figure 11. Example of eDrive.ssd.

This light-weight co-simulation player is easy to extend for the pursuit of 'X-In-the-Loop' methodology. The 'X' stands for 'model'', 'software'', 'hardware', and 'human'. By connecting various abstract models and devices, our FMI/SSP virtual system would be widely expanded. With the flash-based integration of co-simulation browser, users can easily access mobile simulations and visualizations of FMI models in the cloud environment. The FMI co-simulation slaves would be executed on network distributed servers. The control of co-simulation master can be included through the brand-new smart devices with some intuitive multi-touch operations.

The co-simulation browser was applied to check the project example of eDrive.ssd test case. The XML parser reads the FMU connections and interprets the FMI-compliant simulation parameters. The layout of FMU slaves is automatically adjusted in a circular configuration to show the complex connections as compact as possible (see r.h.s in Figure 12). Before starting the system-level simulation, the co-simulation browser can run the unit tests of each FMU with manually added test I/O functions to fit into the FMI input ports.

The system parameter dataset/database could also be distributed on the network servers. There is a process integration tool Optimus® that can export a parameter database wrapped as a portable FMU. For example, the parameters compiled as ResponseSurfaceModel.fmu is

easily imported to our co-simulation test bed, namely, 'FMI/SSP Stage'.

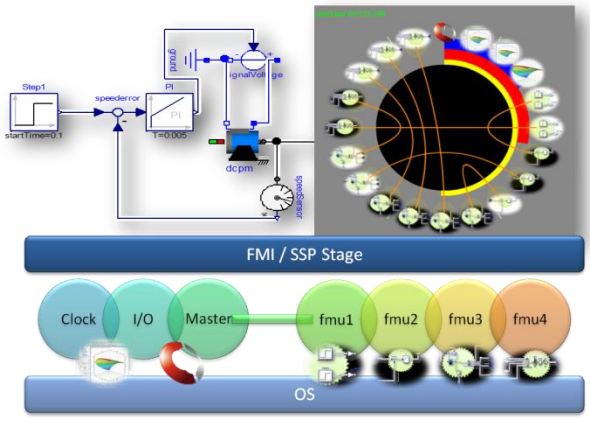

Figure 12. Connection UI on FMI/SSP Stage.

The FMI/SSP stage master manages the simulation context which contains time integration scheme, simulation clock, event handling, and parameter input/output functionalities. In a master-slave co-simulation, the master algorithms can define the quality and performance of the co-simulation. The clock functions setup the master timestep sizes to optimize the simulation efficiency. The way how to share such simulation parameters would be discussed further in the ongoing SSP project.

When we complete the definition of System Structures and System Parameters on this SSP test bed, we can extend the co-simulation environment, to that of 'Co-Optimization'. A FMI co-optimization case with the sequential tool chain process is reported (Batteh *et al*, 2015). The Co-Optimization stands for the paradigm such that every model is gathering as FMU modules on the virtual system stage of SSP. The simulation result could be reflected onto the system parameter set such as Response Surface Model to refine the next co-simulation trial in the optimization or calibration cycle. The SSP-based environment will enhance the co-optimization paradigm and speed up the parameter exploration in virtual systems.

5.3 FMI Bench

FMI Bench is a product of PMSF IT Consulting that provides a workbench for manipulating and integrating FMUs into assembled systems which can then be exported as new complex FMUs for use in other simulations or complete executable simulations for stand-alone use.

As part of the work on SSP a prototypical implementation of the SSP drafts has been undertaken, allowing the importation and exportation of complete SSP packages from FMI Bench.

Special consideration was placed on the ability of SSP to describe systems with external interfaces that would allow exportation as complex FMUs so that systems packaged as SSPs could be re-exported as complex FMUs for use as subsystems in other simulation systems while still making use of the FMI Bench features, such as automatic multi-threading of complex FMUs or remote FMU execution.

The experiences with the SSP drafts showed that this usage was indeed possible, as seen in Figure 13, showing both the imported eDrive example SSP in the upper window and the generated native FMI Bench project, which allows direct code-generation, in the lower window.

Future work is intended to track the progress of the SSP project work in the areas of parameters and complex communication primitives, while integrating SSP/SSD support into the core product.

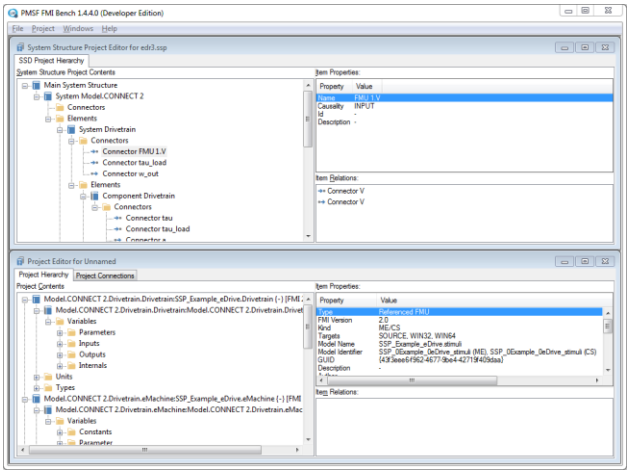

Figure 13. eDrive example in the FMI Bench SSP prototype showing imported SSP and derived native FMI Bench project.

6 Outlook

As shown SSP is a valid approach to make the work with FMUs and their parameterization easier especially when complex systems with several components have to be simulated and interchanged. The way to define system structures is derived from daily work in industry so it can be easily adapted to existing working processes. The close cooperation with the FMI project group guarantees, that both standards work well together, even if SSP is not solely restricted to working with FMUs as components.

In the first stage the project group concentrated on defining system structures. That work is currently mature enough to enable first practical evaluations of it. Parameterization is the second stage to go into in more detail in 2016. The overall goal is to have a first version of the standard rather early to be able to get experience quickly by evaluating it with running prototypes which are developed in parallel. Therefore it is very appreciated if many tool vendors and key users contribute to the project. If you are interested in more information or if you want be get involded in our work,

feel free to contact us: `map-ssp@modelica.org` [Maybe add contact information to SSP working group for vendors/users to join].

References

Blochwitz T., Otter M., Arnold M., Bausch C., Elmqvist H., Junghanns A., Mauß J., Monteiro M., Neidhold T., Neumerkel D., Olßon H., Peetz J.-V., Wolf S., Clauß C. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. *Proceedings of the 8th International Modelica Conference*, pp.105-114, Dresden, Germany, 2011 doi:10.3384/ecp11063105.

Batteh J., Gohl J., Pitchaikani A., Duggan A., Fateh N. Automated Deployment of Modelica Models in Excel via Functional Mockup Interface and Integration with modeFRONTIER. *Proceedings of the 11th International Modelica Conference*, pp.171-180, Versailles, France, 2015 doi:10.3384/ecp15118171.