Introducing the Virtual Systems Interface for Dynamic Coupling of Continuous Time Systems with Discontinuities

Jeffrey Morgan[1] Bruno Loyer[2]

[1]General Motors, United States of America, jeff.morgan@gm.com
[2]Siemens Digital Industries Software, France, bruno.loyer@siemens.com

Abstract

This paper introduces the Virtual Systems Interface (VSI) as a potential enhancement of the FMI interface or as a separate open source interface. The VSI interface is intended to simplify model exchange and dynamic model coupling of continuous time systems with discontinuities while ensuring fast, accurate and low-cost simulations. The VSI interface uses a single interface for the coupling of both continuous time and discrete time systems. It is designed for variable time step integration with proper discontinuity handling and convergence checking. It requires no run-time licenses for any model packaged using the interface.

Keywords: virtual systems interface, dynamic coupling, controls integration, functional mock-up interface

1 Introduction

Automotive and other industries are relying on the use of simulation models to reduce product development time and cost. This has generated a need for the dynamic coupling of both physical system models (called plant models in this paper) and electronic controller models. Furthermore, many systems are composed of subsystems and components produced by different companies so the exchange of models and their integration into simulation packages is required. This has led to the development of the Functional Mock-up Interface (FMI). The Function Mock-up Interface is an open specification which allows import and export of both plant and controller models into and out of any simulation packages that support the specification.

There are two types FMI interfaces: model exchange and co-simulation. The model exchange interface is intended for use with models that are described by differential, algebraic and discrete equations with or without discontinuities, and the system equations are solved simultaneously. The co-simulation interface is intended for use with models where each subsystem is solved independently, and data is exchanged between subsystems only at discrete communication points. In general, the co-simulation interface gives accurate results only for systems that physically exchange information at discrete communication points (i.e. digital controllers) or have low degrees of dynamic coupling so that the destabilizing effect of the discrete communication can be made negligible by selecting a small enough communication interval.

This paper focuses on the model exchange interface and how it can be used effectively for the dynamic coupling of continuous time systems with discontinuities. First, two example usages of the FMI 1.0 model exchange interface are presented, and the issues encountered are explained. Next, some background for effective usage of a model exchange type interface is presented. Finally, a new interface is introduced which is intended to ensure fast, accurate and low cost means for dynamic coupling of continuous time systems with discontinuities.

2 FMI 1.0 Model Exchange Interface Examples and Issues

Two examples are presented using the FMI 1.0 model exchange interface. The first example is an automotive driveline as an example of a continuous time plant model with discontinuities. The second example is an engine dynamic model as an example of a continuous time behavioral controls model with discontinuities. The results and the issues encountered are explained.

2.1 Example 1: Automotive Driveline

The first example is an automotive driveline torsional model. This model is incorporated into several vehicle systems models for the evaluation of both the vehicle hardware and control system designs. The automotive driveline model contains simple torsional elements (springs, dampers, inertias, clutches, planetary gear sets, etc.). The model is run in fixed gear with the torque convertor open, so there are no active controls. The automotive driveline model is a continuous time system with discontinuities. The discontinuities occur due to the inclusion of backlash elements in the model. The automotive driveline model is shown in Figure 1.

The automotive driveline model is implemented in Simcenter Amesim version 15.1 using Microsoft Visual Studio 2010 64 bit as the compiler. The Simcenter Amesim FMU generation tool was used to export the model to FMI 1.0 model exchange format and then it was imported back into Simcenter Amesim as an FMU to verify that it runs correctly.

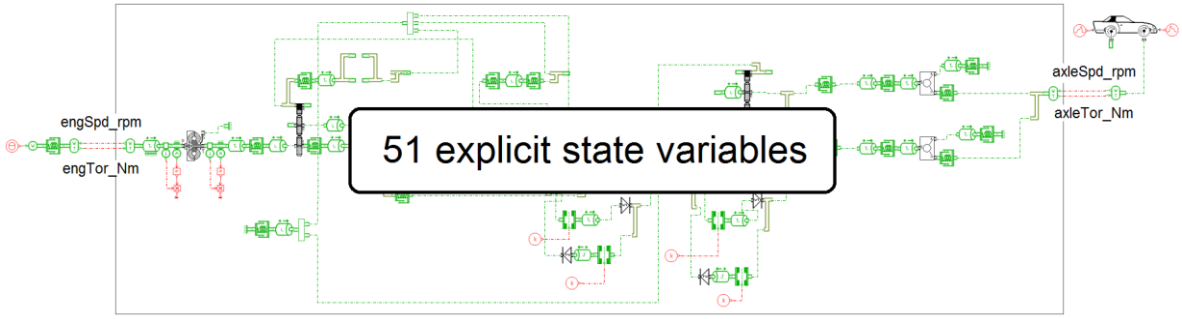

Figure 1. Automotive Driveline Torsional Model (Simcenter Amesim model)

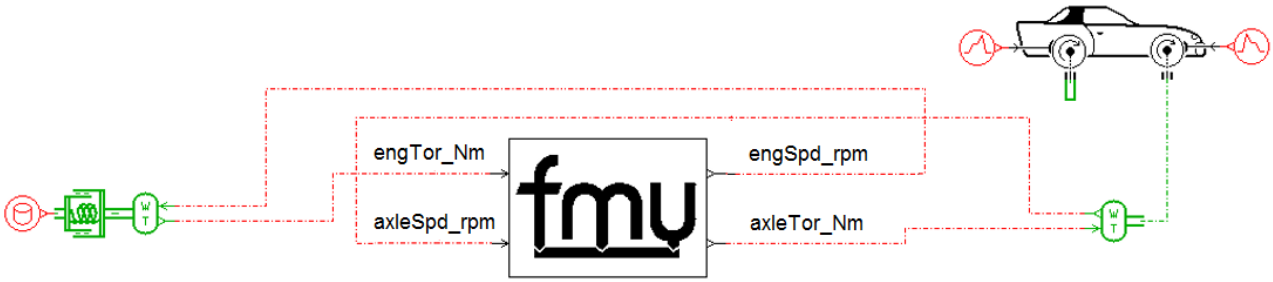

Figure 2. Automotive Driveline FMU integrated into a Vehicle System Model (Simcenter Amesim model)

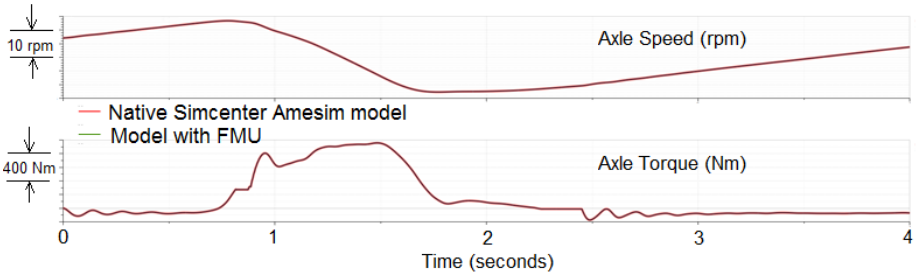

Figure 3. Results for Native Simcenter Amesim Model vs. Simcenter Amesim Model with FMU

The automotive driveline FMU is integrated into a vehicle system model as shown in Figure 2. The simulation was run using the Simcenter Amesim standard integrator and a 1 ms print interval. The results for the system simulation with the automotive driveline included as native elements and as an FMU is shown in Figure 3 (results are the same and show up as one trace).

The FMU produced accurate results but was 10.3 times slower than the native Simcenter Amsim model (24.7 seconds vs. 2.4 seconds). The slow simulation was determined to be caused by the generation of an implicit variable when connecting the FMU. This artificial algebraic loop can be avoided by a better handling of the model's input-output dependencies when exporting the FMU. It should be noted that improving the management of these artificial algebraic loops was one of the motivations behind FMI 2.0.

2.2 Example 2: Engine Dynamic Model

The second example is an automotive gasoline engine dynamic model. This model is incorporated into several vehicle systems models for the evaluation of both the vehicle hardware and control system designs. The engine dynamic model was built in Simulink and a 3rd party commercial FMI generation tool was used to export it to FMI format so that it could be used in other simulation packages. The engine dynamic model is a continuous time system with discontinuities. The discontinuities occur due to the inclusion of continuous time rate limiter elements in the model. The engine dynamic model is shown in Figure 4.

The model is implemented in Simulink version 2014a (64 bit). FMUs were generated using Modelon software version 2.6.1 and then imported into Simcenter Amesim version 14.2 using Microsoft Visual Studio 2010 64 bit as the compiler for system simulation. Even though the Model Exchange interface was used, it was found that the exported model was different depending on which Simulink solver was specified at the time of export. When a fixed time step solver (ODE4 with 1 ms time step) was specified the exported FMU was generated with each time step being a discontinuity (call this the

DOI 10.3384/ecp2017413

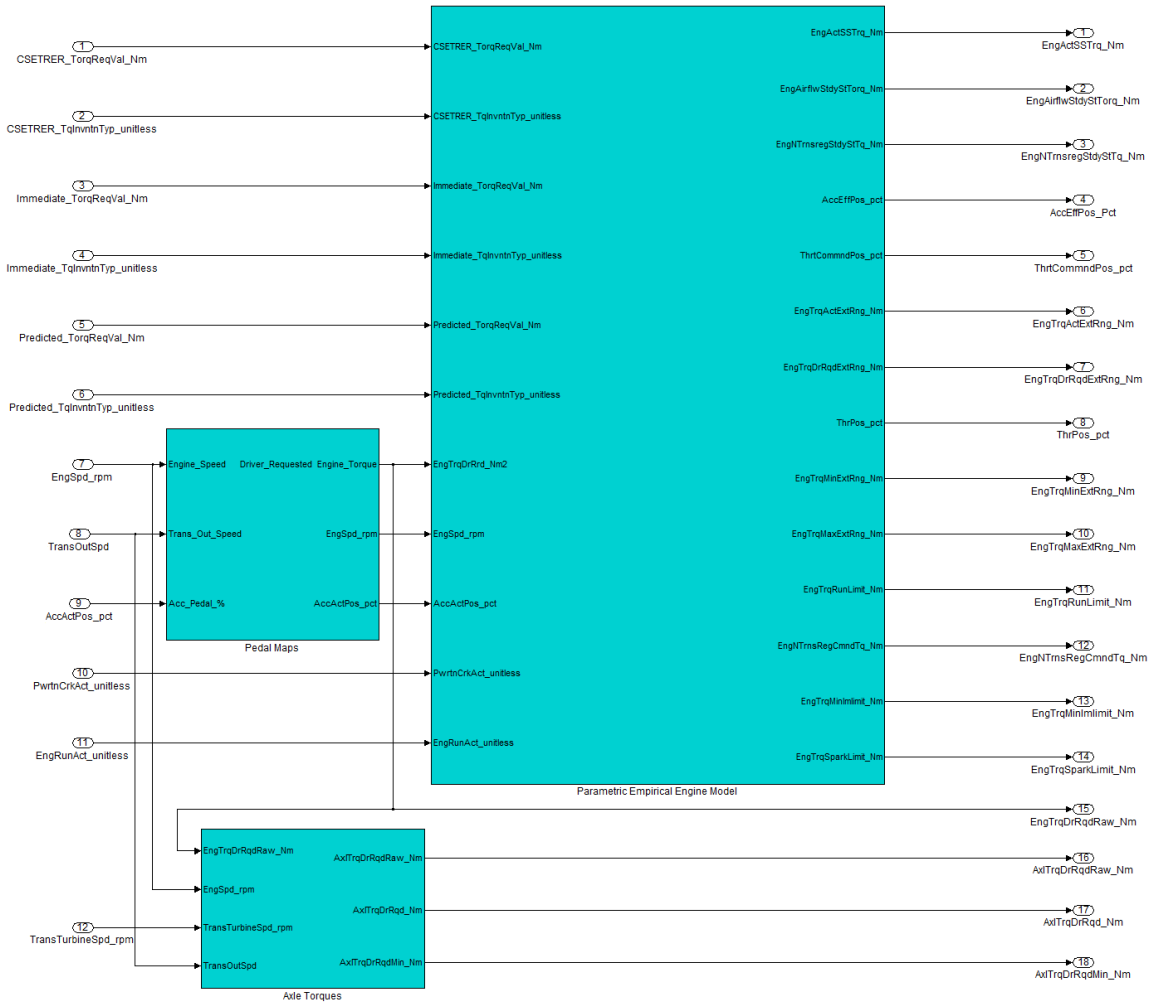

Figure 4. Engine Dynamic Model (Simulink Model)

FMU with time-based discontinuities). When a variable time step solver (ODE45 using a relative tolerance of 1e-3) was specified the exported FMU was generated without each time step being a discontinuity but including event-based discontinuities (call this the FMU with event-based discontinuities). For comparison, a Simcenter Amesim equivalent model was created so the results could be compared. The results are shown in Figure 5.

The simulation was run using the Simcenter Amesim standard integrator and a 1 ms print interval. The FMU with time-based discontinuities produced accurate results while the FMU with event-based discontinuities did not. The FMU with time-based discontinuities was 65 times slower than the native Simcenter Amesim model (1733 seconds vs. 26 seconds). The FMU with event-based discontinuities was 1.3 times slower than the native Simcenter Amesim model (33 seconds vs. 26 seconds). The slow simulation speed of the FMU with time-based discontinuities was due to the creation of discontinuities at each time point corresponding to the Simulink model's originally specified integration time step (1 ms). The inaccuracy of the FMU with event-based discontinuities was due to incorrect handing of the model's real discontinuity points.

2.3 FMI Model Exchange 1.0 Interface Issues

The learnings from the two examples was that the FMI 1.0 Model Exchange interface does not contain enough information on how the internal equations, discontinuity handling, and solution algorithms should be implemented in order to ensure that fast and accurate FMUs are generated.

This paper proposes a new (extended) interface intended to address these issues. The new interface combines Virtual Systems in-the-Loop (VSiL) technology developed jointly by General Motors with the support of LMS (now Siemens Digital Industries Software) and Autonomie technology developed by Argonne National Labs. These two technologies are described next.

3 Virtual Systems in-the-Loop (VSiL)

Virtual Systems in-the-Loop (VSiL) has been used at General Motors since 2005 (Glaue, 2006). This

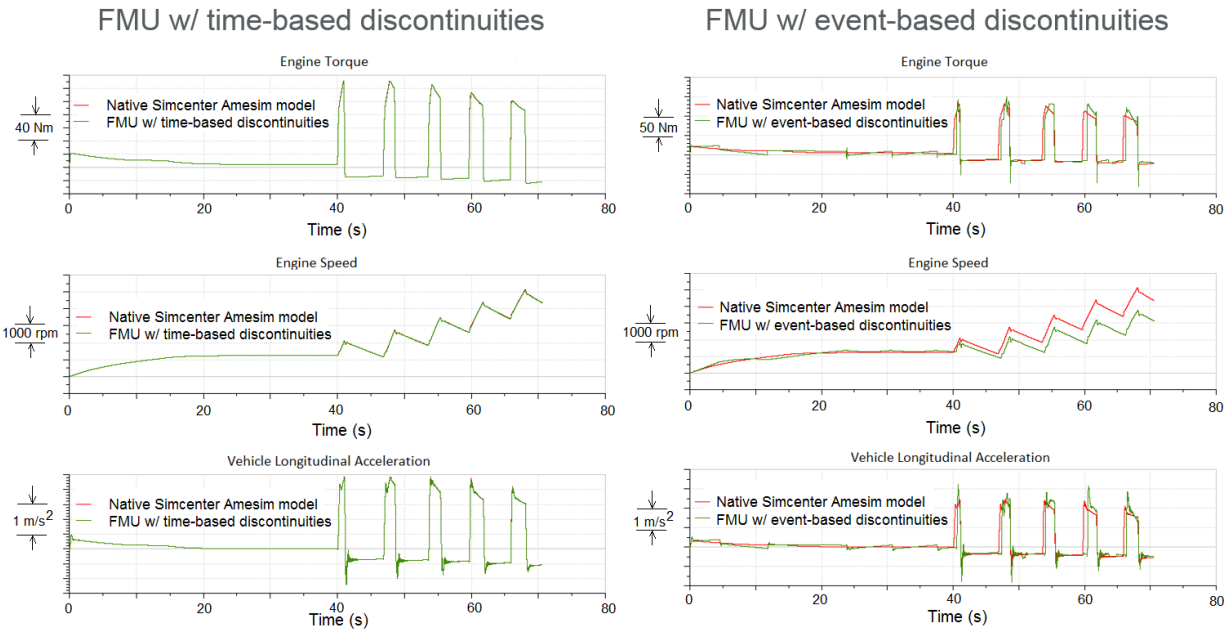

Figure 5. Results for the Engine Dynamic Model Represented Using FMI 1.0 Model Exchange

methodology allows electronic control units (ECUs) to be integrated with Simcenter Amesim plant models and run as a single executable. The Simcenter Amesim plant model is modeled using standard and custom sub models based on Bond Graph (powerflow) methodology (Rosenberg and Karnopp, 1983). This approach defines causality between the powerflow variables which determines how the equations are assembled and solved.

The controller model is built using the actual software source code corresponding to the application layer. The application layer includes all the control algorithms and calibrations which are independent of the actual controller hardware. A hardware input/output (HWIO) layer is built to replace the actual controller hardware specific software which includes all other software code needed to run the ECU, but which is not included in the application software code (i.e. the hardware input and output drivers, task scheduler (RTOS), CAN messaging, etc.). The application code is compiled into an executable file which connects to the plant model thru the glue layer. The glue layer includes method declarations for all the external methods and variables referenced by the application code and any default definitions used when only partial ECU functionality is being analyzed. A set of Simcenter Amesim custom submodels is created to implement the sensors, actuators, scheduler, CAN messaging and peek-inside-ECU functionality. An example engine camshaft phaser VSiL controller model is shown in Figure 6.

Sensors read continuous time signals in engineering units from the plant model and convert these into controller variables defined by the HWIO interfaces. Actuators take controller variables defined by the HWIO interfaces and convert them into continuous time signals in engineering units which are sent to the plant model. The scheduler determines when controller

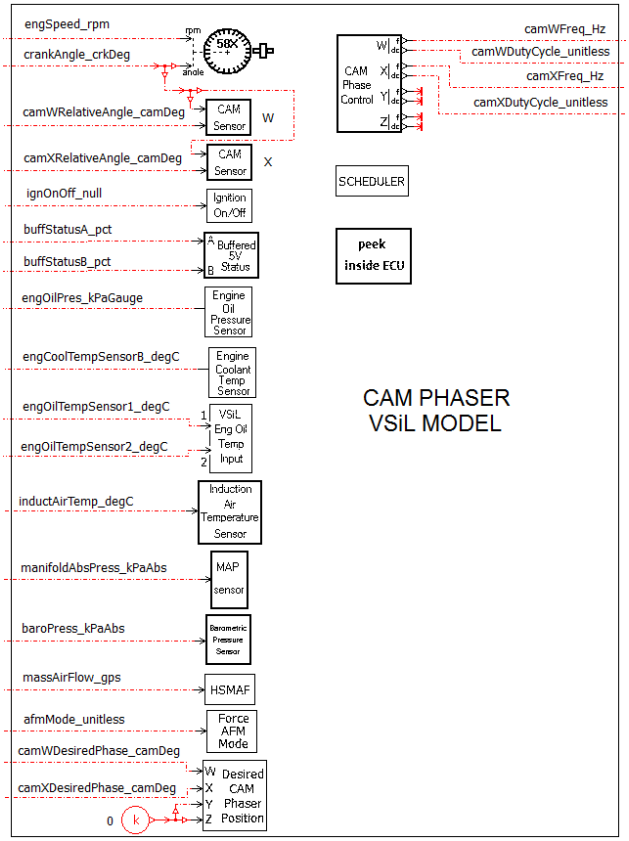

Figure 6. Engine camshaft phaser VSiL controller (Simcenter Amesim model)

methods are executed including controller initialization and the updating of a limited set of calibrations. The CAN messaging sub models handle inter-controller communication signals. The peek-inside-ECU sub models expose a limited set of internal ECU variables for printing and plotting.

DOI 10.3384/ecp2017413

VSiL is intended for hardware focused analyses where the controls are treated as a black-box with only a limited number of controller functionalities (rings), calibrations and internal variables are of interest. Other approaches are used for fully functional software in-the-Loop (SIL) models where all functionality of the controller is of interest.

4 Autonomie

Autonomie is a software package primarily used to analyze vehicle performance and fuel economy (Halbach *et al*, 2010). Autonomie is based on Bond Graph methodology (powerflow) concepts but is implemented using a signal flow approach. This makes it easy to connect plant models to controller models which are also generally implemented using a signal flow approach. Autonomie uses MATLAB/Simulink as the master simulator and any external systems can be integrated as S-functions. The S-Function interface has both a standard (model exchange) and co-simulation interface type. The system level interface used in Autonomie is shown in Figure 7.

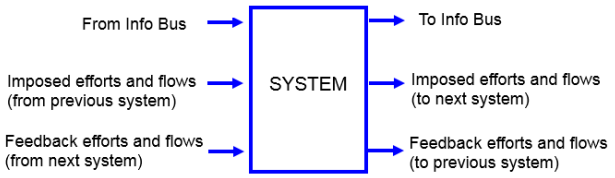

Figure 7. System Level Interface used in Autonomie

The key feature of the Autonomie interface is that it separates the power flow variables from the non-powerflow variables. This concept will be used in the VSI Interface and its importance will be explained later in this paper.

5 Fundamental Problem with Co-simulation Type Interfaces

Before introducing the new interface, we will explain the fundamental problem with co-simulation type interfaces which led to the development of the Virtual Systems Interface. First, consider the operation of a real electronic control unit (ECU) and how data is logged. This is illustrated in Figure 8.

The current time stamp is used for all variables updated during the execution of controller code that is run as a result of a task initiator. This makes it look like these variables are updated at the same time. In reality, there is a time delay between the task initiator and the updated variables.

For the co-simulation interface (parallel execution and fixed time step communication) the execution is done as shown in Figure 9.

The inputs at the previous time step (t_{n-1}) are used to calculate the outputs at the current time step (t_n). Then, the inputs at the current time step (t_n) are updated. We will call this Execution Order 1:

Step Forward > *Update Outputs* > *Update Inputs*

A different execution order can also be used which updates the inputs at the current time step (t_n) first and then calculates the outputs at the current time step (t_n). We will call this Execution Order 0:

Update Inputs > *Step Forward* > *Update Outputs*

Execution order 0 requires sequential (serial) execution of the co-simulating models. During testing of a two-model co-simulation system (one plant and one controller model), it was found that to match the closed loop simulation results, execution order 1 was needed and to match open loop simulation results, execution order 0 was needed. This was a surprising result which

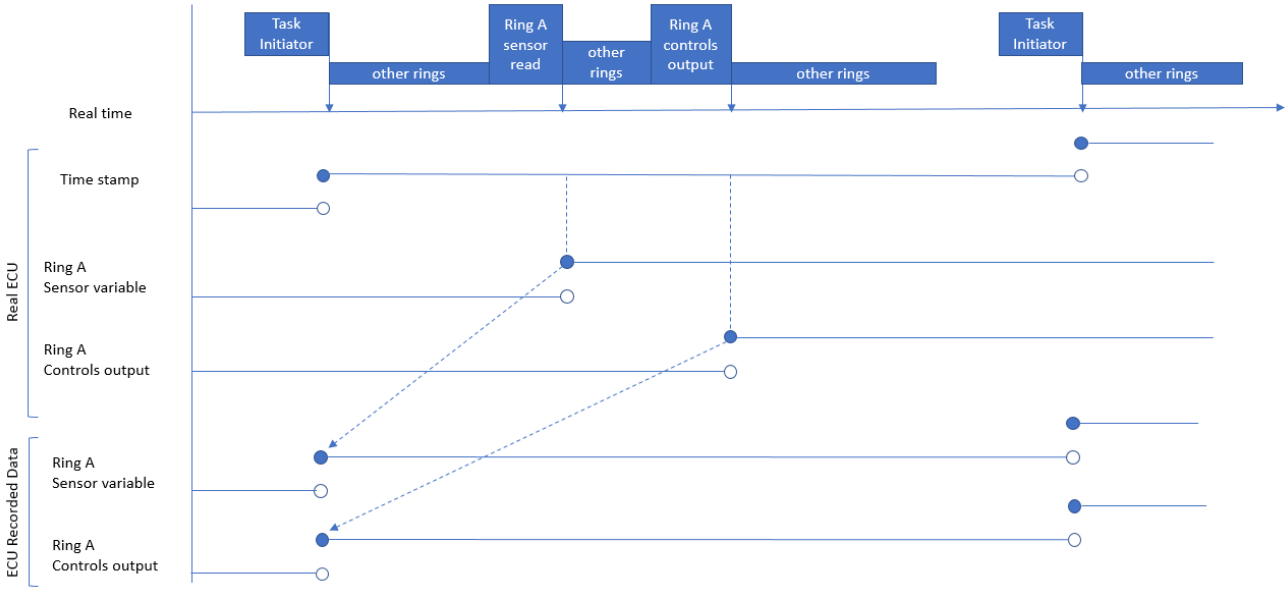

Figure 8. ECU data logging schematic

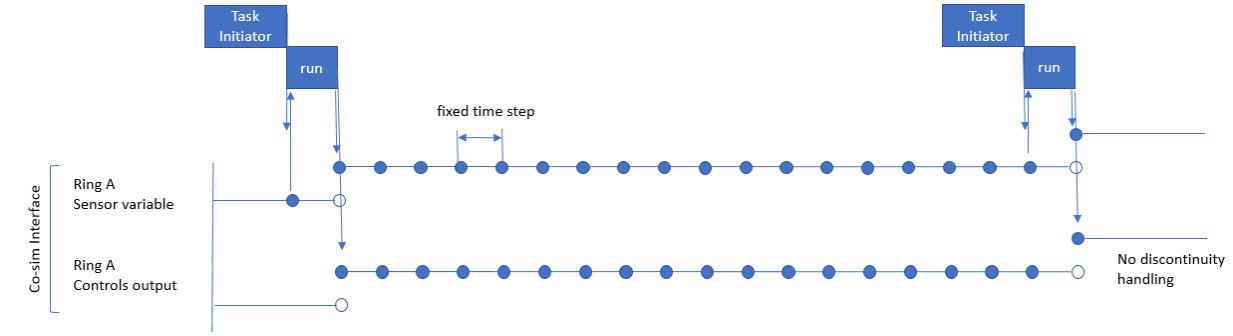

Figure 9. Co-simulation execution schematic

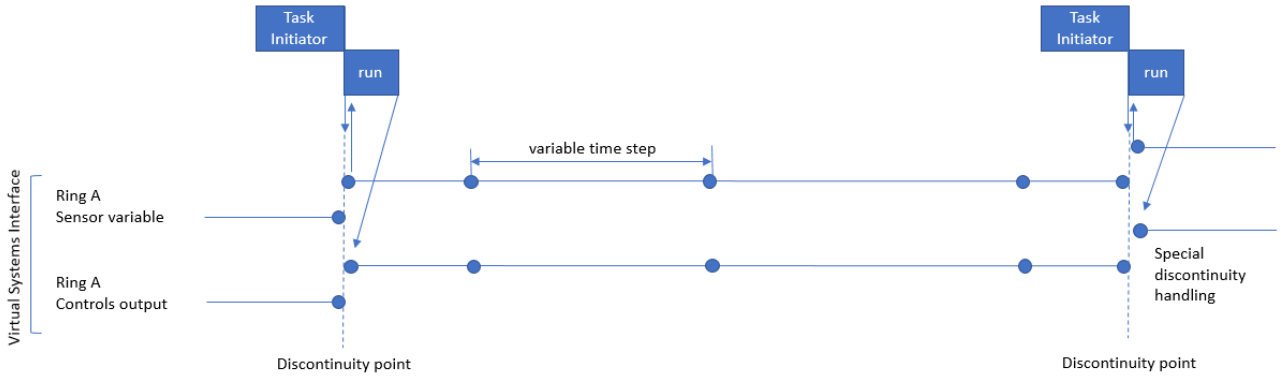

Figure 10. VSI interface execution schematic

led to an extensive investigation into co-simulation data exchange and discontinuity handling. The result of this investigation was that to accurately predict the dynamics of coupled systems (at all coupling strengths and at all time steps) a simultaneous solution of the equations of motion is needed with proper discontinuity handling. The lack of this feature is the fundamental problem with co-simulation type interfaces and was the motivation for the development of the Virtual Systems Interface.

6 Virtual Systems Interface Execution Schematic

The VSI interface has simultaneous solution of the equations of motion and proper discontinuity handling. This is shown in Figure 10.

Discrete systems (like ECUs) are treated as continuous time systems with discontinuities. The VSI interface does not have the sensing and controls delays that are present in actual ECUs (compare Figure 8 and Figure 10). These delays are either neglected or can be added to the sensing and/or actuation part of the plant model (i.e. as continuous time delays or first order lags). These delays are often tuned using current or voltage signals measured directly on the physical wiring of the ECU (not using the ECU recorded variables which do not contain these delays – see Figure 8). The VSI interface is functionally equivalent to the VSiL approach previously described.

7 Virtual Systems Interface Description

The Virtual Systems Interface (VSI) is being introduced as a potential enhancement of the FMI interface or as a separate open source interface intended to simplify model exchange and dynamic model coupling while ensuring fast, accurate and low-cost simulations. The enablers for these features are:

1. Single interface for coupling both continuous time and discrete time systems (for simplicity)
2. Designed for variable time step integration with proper discontinuity handling and convergence checking (for fast and accurate results)
3. Requires no run-time licenses for any model packaged using the interface (for low cost)

The proposed Virtual Systems Interface (VSI) is shown in Figure 11.

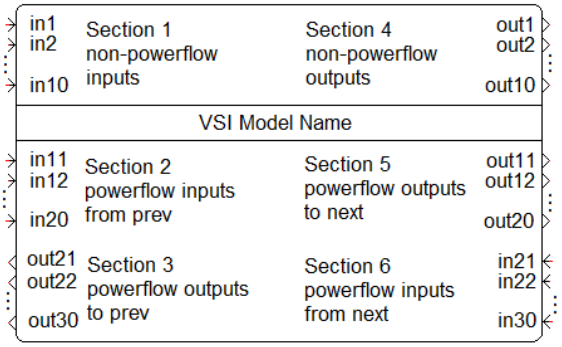

Figure 11. VSI interface Format

DOI 10.3384/ecp2017413

The VSI interface is similar to the Autonomie interface except that the location of the power flow ports is revised to have the I/O relative to the previous system and the I/O relative to the next system to be on the same sides. This is done to allow a more direct connection between upstream and downstream models and easier identification of the power flow signal pairs and corresponding causality. The non-powerflow variables are located above the VSI Model Name and the powerflow variables are located below the VSI Model Name. The inputs and outputs are numbered top to bottom as shown.

To simplify the connection to other models, all of the I/O signals are double precision continuous time variables. Any conversion to discrete time variables is done internal to the interface.

The I/O signals in section 1 and 4 of the interface are used for non-powerflow variables. Both the inputs and outputs are piecewise continuous, and the interface includes a mechanism to report the location of discontinuities so that these are handled correctly. There can be state variables that are integrated internally and/or state variables that are sent to the external integrator for integration.

The I/O signals in section 2, 3, 5 and 6 of the interface are used for the physical coupling of the models using the power flow concept from Bond Graph methodology. The power flow approach requires that all signals be done in pairs where one signal is an effort type variable (force, torque, voltage, pressure, thermodynamic temperature, etc.) and the other signal is a flow type variable (linear velocity, angular velocity, current, volume flow rate, entropy flow rate, etc.). The multiplication of the two paired signals give the input or output power. The power flow signals from Section 2 and 3 of the interface define the input power to the model. The power flow signals from Section 5 and 6 of the interface define the output power from the model. The sign convention for the power flow signals is defined as shown below:

Output Power = Input Power + Internal Power Generation

Internal power generation is negative for internal power loss. Since Bond Graph methodology is used, these signals produce ordinary differential equations (ODEs) with explicit state variables only. The interface includes a mechanism to report the location of discontinuities to the external integrator so that these are handled correctly.

Both continuous time and discrete time systems can be modeled using the VSI interface. A discrete time system is defined as any system that includes one or more discrete time components.

8 Master Algorithm

The VSI interface tries to re-use as much of the FMI nomenclature and programming specifications as possible. When a model is compiled using the VSI interface it is called a VSU (Virtual Systems Unit). The VSI interface is designed to use a single master algorithm that implements variable time step communication and/or integration. The master solver high level flow chart is shown in Figure 12.

A variable time step master algorithm was selected to ensure the fast and accurate solution of the system equations. The VSI interface is not intended for use with fixed time step communication/integration algorithms as these do not handle discontinuities properly.

The VSI interface requires that all components of the system (including each VSU) be continuous time systems with or without discontinuities. Any discrete time elements must be encapsulated inside the VSI interface along with their corresponding analog-to-digital and digital-to-analog convertors. This approach is specifically done so that digital controllers can be

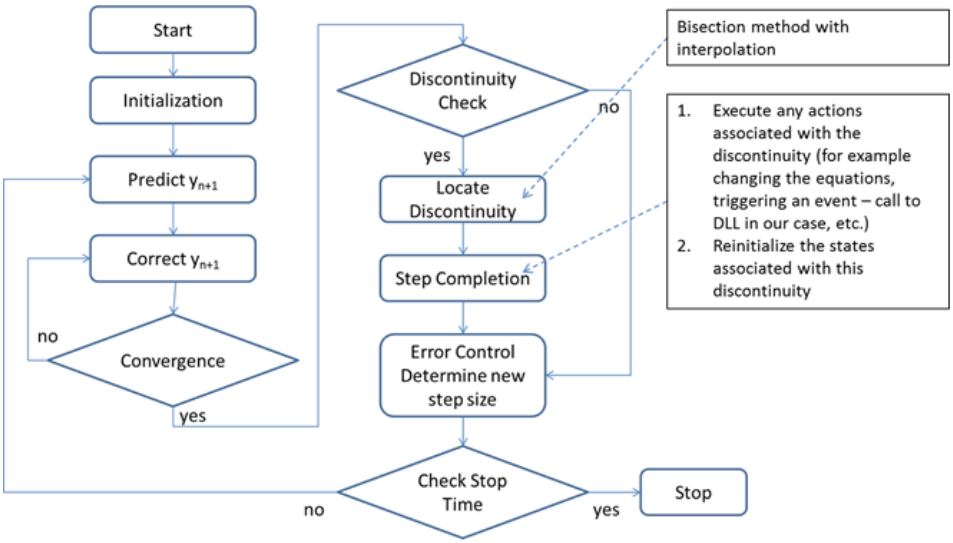

Figure 12. VSI Interface Master Solver High Level Flow Chart

packaged into VSUs and coupled to continuously time plant models using physical signals corresponding to force, torque, voltage, pressure, thermodynamic temperature, linear velocity, angular velocity, current, volume flow rate, entropy flow rate etc. as opposed to the sensed electrical versions of these signals. This approach allows the HWIO layer of the digital controller to be implemented in varying levels of fidelity inside the VSU without having to change the plant model interface. Low fidelity HWIO layers would be typical of ideal sensing (no errors). Medium fidelity HWIO layers would be typical of sensing with empirical models of the sensing errors (bias errors, random errors, first order delays and lags, etc.). High fidelity HWIO layers would be typical of physics-based models of the actual sensors and actuators with state variables that are either solved internally or externally (or combination of both).

To be considered compliant with the VSI specification, the master solver must implement the flow chart shown in Figure 12 and provide a PECE integrator based on Heun's method (Dobrushkin, 2014) as a user selectable option. Having a common integrator option is done to provide a means to validate and compare different venders' master solvers. It is allowed and encouraged that each venders' simulators contain more advanced integration options. It is expected that some systems will not be able to be solved with some integrator types so having either a manual or automatic selection of the integrator is desired. This will automatically encourage venders to create and optimize their master solvers so they can effectively handle a wide variety of equations types. Since the master solvers are expected to contain proprietary implementations, licensing of the master simulator is allowed.

The VSI interface is designed to produce the same set of ODEs regardless of which external integrator (and simulation environment) is used. It is also expected that the same results will be generated regardless of which external integrator (and simulation environment) is used (provided convergence at each time step). This was not the case with FMI 1.0 model exchange interface.

The required features of the VSI interface are shown in Table 1 along with a comparison to FMI 2.0.

Table 1. Required Features of VSI

Required Features of VSI	Part of FMI 2.0 Model Exchange Interface?
Register discontinuities	Yes
Indicate a passed discontinuity	Yes
Reinitialize after a discontinuity	Yes
Indication that a time step is converged	Theoretically feasible
Indication that a time step is used for printing	No
Separation of power flow and non-power flow signals in the interface	No
Restriction of input and output signals to be double precision variables	No
Standardization on how the internal equations are implemented and solved	No

The FMI 2.0 specification already contains some of the VSI interface features required. Thus, the VSI interface can be thought of as an abstract super-class (objective oriented programming terminology) of the FMI interface.

9 VSI Interface Examples

Five examples are presented to illustrate typical use case types for the VSI interface. These examples are described in Table 2.

Table 2. VSI Interface Examples

#	Example Name	Has Power Flow Signals?	Has states requiring external integration?	Has discontinuities?
1	Translational Mass	Yes	Yes	No
2	Dynamic Rate Limiter	No	Yes	Yes
3	Automotive Driveline	Yes	Yes	Yes
4	Behavioral Engine Dynamic Model	No	Yes	Yes
5	Engine Camshaft Phaser VSiL Controller	No	No	Yes

9.1 Example 1: Translational Mass

The first example is a one degree of freedom translational mass with power flow connections on both sides. This shows how simple components can be implement using the VSI interface. The free-body diagram of the translational mass is shown in Figure 13. The representation using the VSI interface is shown in Figure 14.

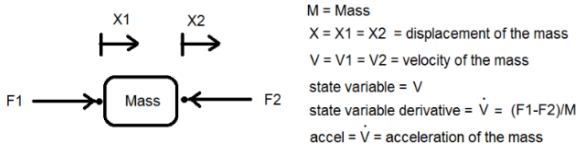

Figure 13. Free Body Diagram of a 1- DOF Translational Mass

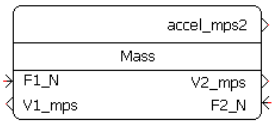

Figure 14. 1-DOF Translational Mass VSI Interface block

The state variable *V* is integrated using the external integrator using the supplied initial condition and state variable derivative value. Causality of the signals is clearly shown by the I/O nature of the signals. Forces are imposed on the mass and velocities are returned. Input power is *F1* times *V1* and output power is *F2* times *V2*. Acceleration of the mass was implemented as an output signal, but it could have been kept as an internal variable used for printing purposes only.

9.2 Example 2: Dynamic Rate Limiter

The second example is a dynamic rate limiter. This is an example of a continuous time system with discontinuities in the state variable derivative and no power flow signals. The functional description of the dynamic rate limiter is shown in Figure 15. The

DOI 10.3384/ecp2017413

representation using the VSI interface is shown in Figure 16.

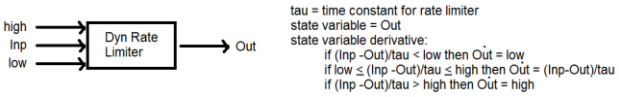

Figure 15. Functional Description of the Dynamic Rate Limiter

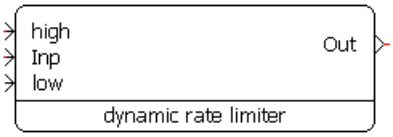

Figure 16. Dynamic Rate Limiter VSI Interface Block

The state variable *Out* is integrated using the external integrator using the supplied initial condition and state variable derivative value. A discontinuity is reported to the external integrator whenever the state variable derivative is discontinuous, and the external integrator takes the proper action. The numerical method used to identify the discontinuity point (time) can be any suitable type but the time corresponding to the discontinuity point must be between the last converged time step and the time step which identified the discontinuity. One approach of discontinuity handling is to linearly extrapolate the discontinuous equation and interpolate the time corresponding to the discontinuity as shown in Figure 17 for the dynamic rate limiter. The FMU must report only real discontinuities to ensure fast execution.

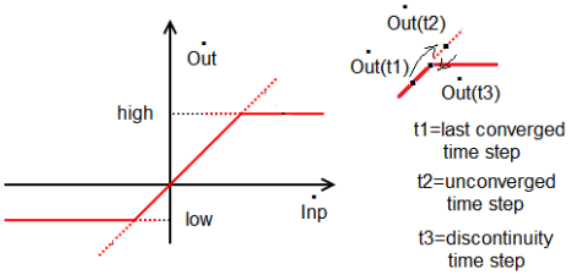

Figure 17. Discontinuity Handling Example

9.3 Example 3: Automotive Driveline

The third example is an automotive driveline torsional model. This is an example of a continuous time system with discontinuities and power flow signals. The automotive driveline torsional model was previously shown in Figure 1. The representation using the VSI interface is shown in Figure 18.

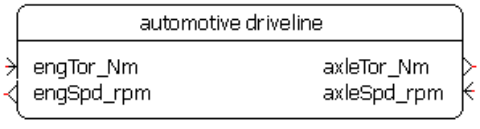

Figure 18. Automotive Driveline VSI Interface Block

The state variables are integrated using the external integrator using the supplied initial conditions and state variable derivative values. Causality of the signals is clearly shown by the I/O nature of the signals. The automotive driveline is modeled in a fixed gear state so there are no control signals sent to or from the model.

9.4 Example 4: Engine Dynamic Model

The forth example is an engine dynamic model. This is an example of a continuous time system with discontinuities and no power flow signals. The engine dynamic model was shown previously in Figure 4. The representation using the VSI interface is shown in Figure 19.

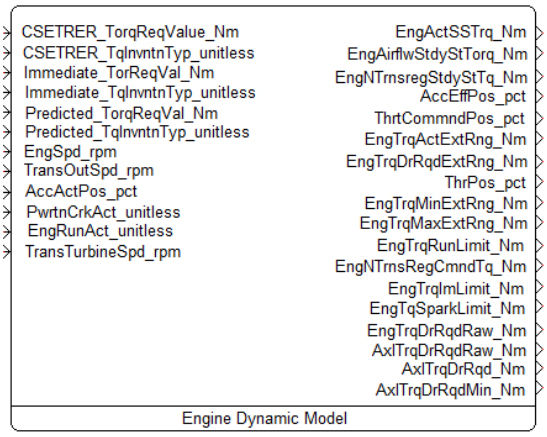

Figure 19. Engine Dynamic Model VSI Interface Block

The engine dynamic model does not have any power flow signal pairs identified in the interface (no signals below the VSI Interface name), but this does not mean there is no power flow. Any system with inputs and outputs can produce power flow bonds either intentionally or unintentionally. The engine dynamic model is intended to provide engine dynamic torque to a downstream system with the downstream system providing the engine speed as feedback. Thus, there is a hidden power flow in this interface. This occurs because of the implementation is done using a signal flow model and not a power flow model. Generally, the same physics can be represented in either a signal flow model or a power flow model and the VSI interface can handle either model type and get the same results provided that all the state variables in the non-power flow part of the interface are exposed to the external integrator. This is the case for the Engine Dynamic Model shown in this section.

9.5 Example 5: Engine Camshaft Phaser VSiL Controller

The fifth example is an engine camshaft phaser VSiL controller. This is an example of a discrete time system with no power flow signals. The VSiL controller model was previously shown in Figure 6. The representation using the VSI interface is shown in Figure 20.

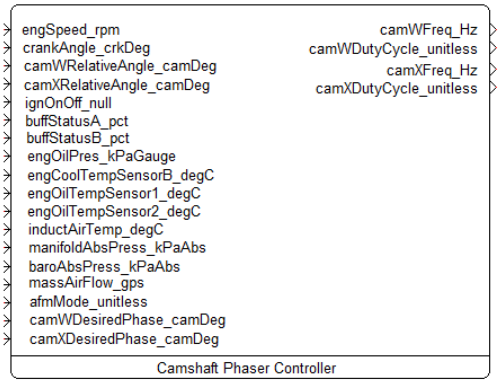

Figure 20. Engine Camshaft Phaser VSIL controller VSI interface block

There are no external or internal state variables to integrate. The camshaft phaser VSIL controller reads the input signals and calculates the output signals whenever the controller detects a time based or event-based task initiator (trigger). The external integrator is notified of each task initiator as a discontinuity point and takes the proper action. The camshaft phaser controller ring does not use any signals from other controllers so no CAN (Controller Area Network) communication signals are included in the VSiL controller.

10 Connecting VSI Interface Models

VSI interface models can be connected to other signal flow models simply by connecting the input and output signals. VSI interface models can be connected to power flow models using ether sensor/actuator pairs or signals to power flow elements as shown in Figure 21.

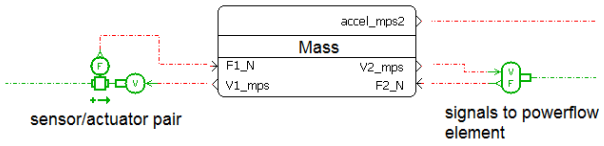

Figure 21. Connection of VSI interface Models

11 Algebraic Loops and Implicit State Variables

The VSI interface is intended to produce the same set of equations as the source models, thus if the source model when combined into a feedback system produces algebraic loops or implicit state variables, the VSU model will likewise produce algebraic loops or implicit state variables. Conversely, if the source model when combined into a feedback system does not produce algebraic loops or implicit state variables, the VSU model will likewise not produce algebraic loops or implicit state variables.

The VSI interface will not solve systems containing algebraic loops or implicit state variables. The VSI interface is designed to handle systems with ODEs (and possibly DAEs in the future).

12 Why the VSI interface Requires No Run-Time Licenses

The reason why the VSI Interface requires no run-time licenses for any model packaged using the interface is so that system simulation cost will be comparable to single component and/or sub-system simulation cost. This also ensures that all models packaged with the VSI interface will run in any simulation package that implements the interface. Software venders can and do charge extra for the FMU generation feature and this is still allowed in the VSI specification.

13 Application Areas for VSI Interface

The intended application areas for the VSI interface are shown in Table 3.

Table 3. VSI Intended Application Areas

Area #	Application Area	Intended for VSI Interface
1	Plant Models for HIL bench	No
2	Plant Models for MIL & SIL	Yes
2	Continuous Time Behavioral Controllers MIL	Yes
3	Discrete Time Ring Level Controllers for SIL & VSiL	Yes
4	Discrete Time Full Controllers for SIL	No

14 Conclusions

The VSI interface is presented in this paper as a high-level overview and will require additional work to formalize its content into a usable specification. Future development of the VSI interface will be dependent on the feedback from the FMI developers and the model coupling and co-simulation user community as to whether these types of enhancements are of significant value for the type of simulations being done for product development in their specific industries.

Acknowledgements

The authors would like to acknowledge the original developers of the Virtual Systems in-the-Loop (VSiL) technology and the Autonomie software which led to the development of the VSI interface.

References

Ronald C. Rosenberg and Dean C. Karnopp. Introduction to Physical System Dynamics, McGraw-Hill, 1983.

Shane Halbach, Phillip Sharer, Sylvain Pagerit, Aymeric P. Rousseau and Charles Folkerts. Model Architecture, Methods, and Interfaces for Efficient Math-Based Design and Simulation of Automotive Control Systems. SAE 2010-01-0241, SAE World Congress, Detroit, April 2010.

Tim Glaue. Virtual Systems-in-the-Loop (VSiL): System Modeling for Quality Controls Integration. 2006 Simcenter Amesim European Users Conference, March 30, 2006.

Vladimir A. Dobrushkin, Applied Differential Equations: The Primary Course, Mathematica Tutorial for the First Course. Part III: Heun Methods, CRC Press, 1st Edition, 2014.