*Proceedings of the 2025 Winter Simulation Conference*
*E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.*

# A METHOD FOR FMI AND DEVS FOR CO-SIMULATION

Ritvik Joshi[1], James Nutaro[2], Gabriel Wainer[1], Bernard Zeigler[3], and Doohwan Kim[3]

[1]Dept. of Systems and Computer Eng., Carleton University, Ottawa, ON, CANADA
[2]Computational Sciences & Eng. Division, Oak Ridge National Lab., Oak Ridge, TN, USA
[3]RTSync Corp., Chandler, AZ, USA

## ABSTRACT

The need for standardized exchange of dynamic models led to the Functional Mockup Interface (FMI), which facilitates model exchange and co-simulation across multiple tools. Integration of this standard with modeling and simulation formalism enhances interoperability and provides opportunities for collaboration. This research presents an approach for the integration of FMI and Discrete Event System Specification (DEVS). DEVS provides the modularity required for seamlessly integrating the shared model. We proposed a framework for exporting and co-simulating DEVS models as well as for importing and co-simulating continuous-time models using the FMI standard. We present a case study that shows the use of this framework to simulate the steering system of an Unmanned Ground Vehicle (UGV).

## 1 INTRODUCTION

Developing modern complex systems often requires integrating expertise from multiple disciplines. Modeling and simulation (M&S) can support such collaboration; however, challenges arise as experts from different disciplines rely on diverse tools and modeling methods. Model sharing can bridge the gap between these approaches, ensuring access to the same information and consistency across multiple groups. With this goal, the Functional Mockup Interface (FMI) standardizes methods for sharing models (Blochwitz et al. 2011). The standard provides methods for model exchange and co-simulation. It streamlines the process of integrating multiple system components using different tools.

One of the model-sharing methods described in the standard is called co-simulation, and it allows the interaction of multiple independent models developed using different modeling methods. It permits a shared model to run independently with other models and synchronize only at specific points. This is useful for simulating models with different formalisms. The ability of FMI co-simulation to handle interactions between diverse models aligns with the flexibility of the Discrete Event Systems Specification (DEVS) (Zeigler et al. 2000), a well-known M&S formalism. DEVS models are modular and hierarchical, and can also be integrated with other formalisms for developing hybrid systems.

Integrating DEVS with FMI enables DEVS models to interact with continuous-time models, facilitating the connection between discrete and continuous dynamics without requiring redefinition of the continuous-time models developed in non-DEVS tools. A DEVS-based simulator can import models and use them to perform integrated simulations. The export and import capabilities also enable sharing DEVS models, promoting collaboration between DEVS tools and encouraging reusability. Nevertheless, existing DEVS tools need to be improved to achieve the interoperability essential for developing complex engineering systems. The tools do not provide standardized methods to synchronize the import and export operations during model sharing (like FMI), and the export capability of DEVS models is underexplored, which limits their potential for integration with other simulation frameworks and tools.

This research addresses this gap by defining a framework that exports and imports strategies for DEVS models with synchronization between the two. This was achieved by creating an importer for the Cadmium tool (Cárdenas and Wainer 2022) and an exporter for the MS4Me tool (Seo et al. 2013). The framework

also bridged tools like MATLAB-Simulink and Open-Modelica with the DEVS-based tool Cadmium with the help of continuous time solvers. These solvers performed the task of simulating the Functional Mockup Unit (FMU) of continuous-time models on the DEVS-based simulator. The framework provides flexibility with the use of tools for exporting continuous-time models by taking advantage of a standardized interface for integrating FMUs. This research shows an approach for utilizing the DEVS framework for tool integration using FMI, contributing to the broader adoption of FMI-based co-simulation techniques.

We defined the mechanisms for these interactions; we show the use of these methods in the M&S of steering control for an Unmanned Ground Vehicle, integrating FMI and DEVS. Subsystems are developed in different DEVS-based tools and shared to enable co-simulation. A continuous time model is combined with a DEVS model through FMI co-simulation. The tools include MS4Me, a DEVS-based tool, for the navigation system. The block was independently tested and exported as a co-simulation FMU. A continuous-time PID controller model was built using MATLAB (Simulink), and its response and parameter tuning were performed before exporting it as a co-simulation FMU. The remaining components were developed in Cadmium, another DEVS tool. Cadmium acted as the host simulator for the co-simulation of the system, coordinating the integration of these two components through the FMI.

## 2 BACKGROUND

Unmanned Ground Vehicles (UGV) operate on the ground and do not have an onboard operator (Hebert et al. 2012), running autonomously or controlled remotely. They are engineered for environments where human presence is risky or impractical. UGVs perform a wide range of applications: civil engineering for labor-intensive, costly and dangerous tasks (Hu and Assaad 2023); construction, hazardous material handling, waste management, etc. Agricultural Unmanned Ground Vehicles (Farella et al. 2024) improve perform tasks like monitoring, harvesting and transportation. UGVs are also used in mining applications (Cherukuri and Shashank 2018), border patrol, surveillance and combat.

Extensive research and development are being conducted for UGVs, in areas like mobility, localization, path planning, and navigation (Ragothaman et al. 2021; Chen et al. 2022). M&S provides a controlled and flexible environment for evaluation, and it allows developers to analyze system response for control actions. Simulating the systems in impractical conditions for real world testing helps in making the design safe, reliable and robust. Engineering projects involving multiple disciplines are often distributed among vendors or research teams. Model sharing and model-based system engineering can help build such complex multi-disciplinary applications by integrating smaller sub-systems (from different domains and developed with different tools). Reusing these subsystem models can save significant effort and integrated M&S refines the system's performance (Inzillo and Carlos 2017) (Zhang et al. 2010).

Understanding the importance of model sharing, M&S tool developers made several attempts to create proprietary interfaces for exchanging models (Blochwitz et al. 2011). Standardization of these efforts was achieved by the development of a standard model interface called FMI (Functional Mockup Interface) (Blochwitz et al. 2011) for tool-independent exchange models and co-simulation. The standard includes a model description schema, model packaging format, M&S methods, and application programming interfaces (APIs). Model sharing is done through a Functional Mockup Unit (FMU), a container for a model, built as an executable package containing the model description, its behavior, and all the associated data.

The standard proposes one API for model exchange and one for co-simulation. FMI for co-simulation tries to couple two or more models built in separate tools for a joint simulation (Functional Mockup Interface Standard 2014). The co-simulation is performed by packaging the model as an FMU along with the solver that simulates the model. These packaged models are interconnected, and a co-simulation algorithm is responsible for synchronization and data exchange between models. The communication between the models is restricted to discrete communication points and the model performs numerical integration independently between the two points by their respective solver (Gomes et al. 2018).

Discrete Event System Specification (DEVS) (Zeigler et al. 2000) is a well-established M&S formalism that allows modeling system where inputs and outputs can be represented as a sequence of events. DEVS uses atomic models to describe the system's behavior and coupled models that describe a composition of

other DEVS models. Cadmium (Cárdenas and Wainer 2022) and MS4Me (Seo et al. 2013) are the tools used in this research. Cadmium is a header-only C++17 multi-platform library. MS4Me is developed by MS4 Systems that allows users to define the system's overall structure using a sequence diagram, which converts into a template DEVS model in Java. A System Entity Structure document is generated from the sequence diagram to represent coupled models and state diagrams represent atomic models.

We want to explore the co-simulation of hybrid and continuous models; for instance, a PID controller (Astrom 1995) combined with DEVS models. The classical PID is popular for UGV control because of its robustness, simplicity for design and integration, and performance (rise time, settling time, and stability). Automatic and manual tuning methods exist in tools like MATLAB(Simulink) (Wang 2020).

Continuous models describe systems as a function of time where time is a continuous value, and represent real-world processes using the continuous dynamic of the systems; thus, they need to be simulated on digital computers by discretizing continuous signals. This is achieved by using solvers that often rely on numerical methods (Cellier and Kofman 2006), e.g. Euler's, Runge-Kutta and others. An alternative for the M&S of continuous time models in DEVS is the use of the Quantized State System (QSS) method (Kofman and Junco 2001), which is based on the concept of state quantization (Zeigler and Lee 1998). These methods perform an event-based discretization of the continuous time equations with a pre-defined threshold called a quantum. The mechanism responsible for this event detection is called a *quantizer*, which defines a threshold and monitors the inputs to detect an event based on the input value and logical conditions to decide whether a significant change has occurred. Multiple well-established QSS methods are available, and each method differs in accuracy and efficiency, such as second order QSS (QSS2) (Kofman, 2002).

Recent research has proposed approaches to integrate FMI with DEVS for M&S of cyber-physical systems. (Müller and Widl 2013) proposed a generic scheme for integrating continuous subsystems into a discrete event system with the help of FMI. A formal way to link the discrepancy between the semantics of heterogeneous modeling formalisms and FMI was examined by (Tripakis 2015). A different approach was introduced by (Camus et al. 2018), which proposed a Multi-agent Environment for Complex-SYstem CO-simulation that employed DEVS wrapping to integrate different co-simulation models. Building on this, (Lin 2021) presented a method for simulating four-variable models using the Java-based DEVS-Suite (Kim et al. 2009) and the models developed using OpenModelica. More recent advancements in this domain are in the work of (Vanommeslaeghe et al. 2024) and (Vanommeslaeghe Van Acker et al. 2020), who proposed a method for integrating DEVS with FMI for simulating embedded platforms. The framework is based on the generic architecture proposed by (Müller and Widl 2013). In our architecture, we are trying to use the concept of QSS to make the next prediction and determine the time for the next prediction.

## 3 DEVS-FMI INTEGRATION

We defined a framework that will allow the sharing of continuous time models using scripting languages like Modelica or MATLAB with the help of FMI standards. The framework has an interface in the form of an importer that will allow integrated M&S of shared models using a DEVS-based simulator (like Cadmium). Discrete event models developed using other DEVS-based M&S tools (Like MS4Me) can also be shared using the FMI co-simulation to be incorporated into the simulation. For simulating the UGV steering control model, we have used the Cadmium (Cárdenas and Wainer 2022) tool to develop an interface that will import or use the shared models and perform a discrete event M&S. The Figure 1 below shows a high-level overview of the framework with the help of a block diagram.

The diagram shows blocks for the DEVS-based tool MS4Me and Modelica-based models in Open-Modelica, used to create continuous system models. Open-Modelica supports FMI 2.0 for Model-exchange and co-simulation. The co-simulation FMU exported by the tool is also used by the next block. This co-simulation FMU consists of a model and a solver to simulate such model. Instead, the MS4Me block represents the definition of DEVS models, which is done using a graphical interface, and then converting the model to Java. The block FMU4-MS4Me (detailed in DEVS model Export) is an interface we defined, which is used to export the DEVS model using FMI to generate the co-simulation FMU illustrated in the next block. This FMU contains the DEVS model and the DEVS simulator as the solver. Then, the block on

the right represents Cadmium and all its sub-components. The FMU wrapper or importer is responsible for dealing with the FMU. For co-simulation FMU, this block will act as a wrapper that will create a model instance in Cadmium and communicate with the model using APIs provided by the FMI library. The DEVS models and Modelica model shown inside the Cadmium block are the FMUs of models created by their respective tools. The continuous time models shared through this process will be simulated with the help of solvers designed in Cadmium. This framework enables the import of multiple FMUs from various tools, allowing them to be connected in any configuration to meet specific model requirements.

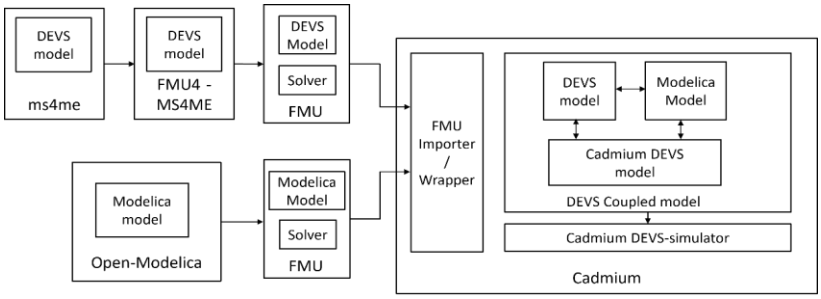


Figure 1: Block diagram for High-level overview of co-simulation in Cadmium.

The FMU importer imports and integrates the model and the FMU4-MS4Me block; these were defined to allow a streamlined way of model sharing. We also defined QSS and Runge-Kutta solvers for continuous models imported to Cadmium. We include the blocks MS4Me and Open-Modelica because these are the tools used for experimentation, but the architecture is generic: any M&S tool capable of exporting an FMU can be used instead of Open-Modelica without modifications. MS4Me can be changed by other DEVS tools, but it would require the FMI export functionality we provide in the FMU4-MS4Me. The continuous time solvers are also generic and allow integration for model simulation even if the exporting tool changes. The framework is scalable as multiple sub-systems can be imported for integrated M&S.

### 3.1 FMI co-simulation

The co-simulation FMU has its solver, which helps execute independently after instantiation. The design requires a strategy to communicate with the instance to get the values of required variables of the model. The functionality was defined following the state machine for co-simulation outlined in (Functional Mockup Interface Standard). By referring to the calling state machine in the standard, the process for the co-simulation FMU follows a sequence of steps – instantiation, configuration, model execution, and termination. Instantiation and configuration use the same APIs as the instantiation and configuration stages in Model exchange with different parameters. In the instantiation function, an instance of co-simulation FMU is created in the atomic model. Information such as model name and model files are provided in the function parameters. The configuration then initializes the variables and sets the initial condition.

The FMU is executed inside the atomic model and updating state variables. The co-simulation FMU can interact with the model only on communication points. It needs the current simulation time and the time for the next communication point to perform variable integration. After receiving the next communication point, FMU integrates the continuous time variables using the solver inside FMU and updates the values. Once the step is completed, the variable values can be accessed and utilized for further computation.

Figure 2 shows a block diagram of the FMI co-simulation defined in Cadmium. The coordination of co-simulation FMUs depends on the co-simulation algorithm. Here, the Cadmium M&S algorithm acts as the co-simulation algorithm. The FMI wrapper block shown in Figure 2, links a leader simulation algorithm and a follower FMU. This is achieved with the help of the APIs developed in the FMU wrapper that can be called inside the atomic models.

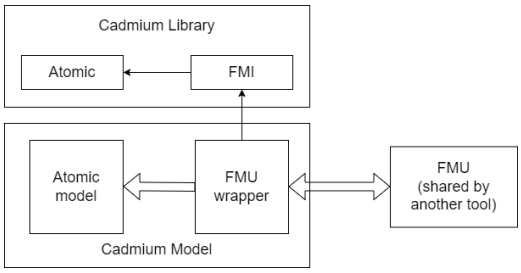


Figure 2: Block diagram of FMI co-simulation in Cadmium.

These APIs facilitate the instantiation and configuration of FMUs, while also managing communication with the FMU to synchronize steps and exchange variable values. The class designed to define these APIs is the same as the FMI importer, as seen in the following code snippet.

```
template <typename S> class FMI: public Atomic<S> {
public:
   FMI (const std::string& id, S initialState, const char* modelname, const char* guid,
            const char* resource_path, const char* so_file_name, double start_time);

    virtual void internalTransition(S& s) const = 0;
    virtual void externalTransition(S& s, double e) const = 0;
    virtual void output(const S& s) const = 0;
    virtual double timeAdvance(const S& s) const = 0;
    virtual void initialize(S& s);
    double get_real(int k) const;
    void set_real(int k, double val) const;
    int get_int(int k) const;
    void set_int(int k, int val) const;
    bool get_bool(int k) const;
    void set_bool(int k, bool val) const;
    void do_step(double CurrentTime, double h) const;
}
```

The FMU instance is created with the help of *fmi2Instantiate()* by passing the *modelname*, *guid*, *resource_path*, and *so_file_name* provided in the model. Model variables are initialized using *initialize()*. The *internalTransition*, *externalTransition*, *output*, and *timeAdvance* functions inherit from the *atomic* class and define the atomic model. All getter and setter functions exchange variables based on their respective data types. The *do_step()* function is used to communicate with the FMU to provide the current execution time and the next communication point for it to perform numerical integration and update variables.

The FMU instantiation and function calls inside the atomic model enable the FMU to connect with the simulation algorithm. The *do_step()* function is called by the model's internal transition function, where the time advance value is used as the next communication point. After the successful execution of the step, the FMU model output and variable values are read using the getter functions, which are then used for further calculations inside the model. These values are then passed through the output ports of the atomic model. The input received via the input port of the atomic model through the external transition function is shared with the FMU. In this way, the FMU is mapped in the atomic model to integrate with the DEVS framework. The DEVS model exported as an FMU is also co-simulated with other DEVS models using this process. In addition, a variable containing the time of the next event is shared as a model variable during the export of the model. This is used to set the time advance of the atomic model communicating with the FMU. As a result, the step function is invoked for each event in the DEVS model within the FMU, as seen in Figure 3.

The Root-coordinator communicates with the Top-coordinator, which receives the $t$ value from the simulator and the coordinator. The value $t$ is the time for the next event. The solver inside the FMU is also sending the $t$ value to the coordinator with the help of the communicating atomic model. We know the root coordinator is responsible for advancing the global simulation time and it advances the time to the lowest time value to simulate the next event.

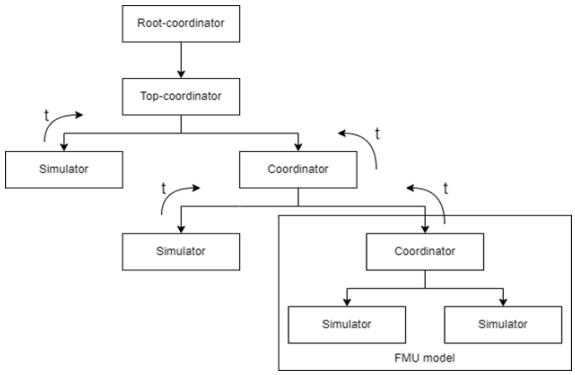


Figure 3: Conceptual representation of FMI co-simulation with abstract simulator.

Setting the time advance value with the above strategy treats the entire DEVS model within FMU as one atomic model. It allows the simulation of all the events of the co-simulating model in the correct sequence according to the global simulation time.

### 3.2 DEVS model Export

Co-simulation in Cadmium was developed to incorporate models from other simulation tools such as OpenModelica. The complementary method for exporting DEVS models as co-simulation FMUs was developed using MS4Me GUI, and converted to Java code. To package this Java model as an FMU, we used FMU4J (Hatledal et al. 2018), a package designed for handling FMUs within the Java Virtual Machine (JVM) environment. This was used to achieve cross-platform compatibility, using FMUs with Java and ensuring interoperability between Java and other programming languages.

Figure 4 shows a UML diagram of the environment defined for exporting the DEVS model as an FMU. FMU4-MS4Me uses the FMU4J library and its functionality to package the models into FMUs. We extended this capability to export DEVS models developed using MS4Me. We designed the functionality by specifying how to define generic DEVS functionality and a second one for model-specific design. The DevsFMU and Application model classes, shown in Figure 4, were defined using the libraries FMU4J and *ms4systems*. The DEVS model package shown in Figure 4 is the model code automatically generated by MS4Me. DevsFMU implements generic DEVS M&S functionality, using Fmi2Helper, which is defined in the FMU4J and supports FMU export. DevsFMU defines objects of the *model* and *simulator* class as its class variables. The *model* class (CoupledModelImpl) is used to describe the model. The *simulator* class (SimulationImpl) contains all the functions required to simulate the model. Both are defined in MS4Me. DevsFMU overrides some of the Fmi2Helper functions: the *registerVariables* function to set and update *timeRemaining*, which defines the time to the next event (using *setupExperiment* to start the simulation). The simulation functions are called by the do_step function, which calls for *injectInput* from MS4Me if it needs to send inputs (otherwise, it calls the *simulateIterations* to perform the next iteration).

DevsFMU provides functions to simulate the DEVS model. To instantiate the model and add model-specific functionalities, we created a new class that extends DevsFMU and configures the variables according to the model's requirements. The model variable is assigned an instance of the DEVS model, and the Simulator is initialized with the simulator design object, which configures the simulation using the model and assigns it a specific name, along with the relevant options. The class for application model also redefines a few functions of its parent class. The *registerVariables* function calls the superclass function that updates *timeRemaining* and registers other variables to be shared. The registration involves declaring the names of the variables being shared, mapping them to the actual state variable inside the model, and setting its properties like causality (input/output), variability (discrete/continuous), etc. The do_step function maps the FMU's input to the model's input to inject an input and then calls the do_step function of the superclass, which terminates the simulation by evaluating the *end-of-simulation* condition.

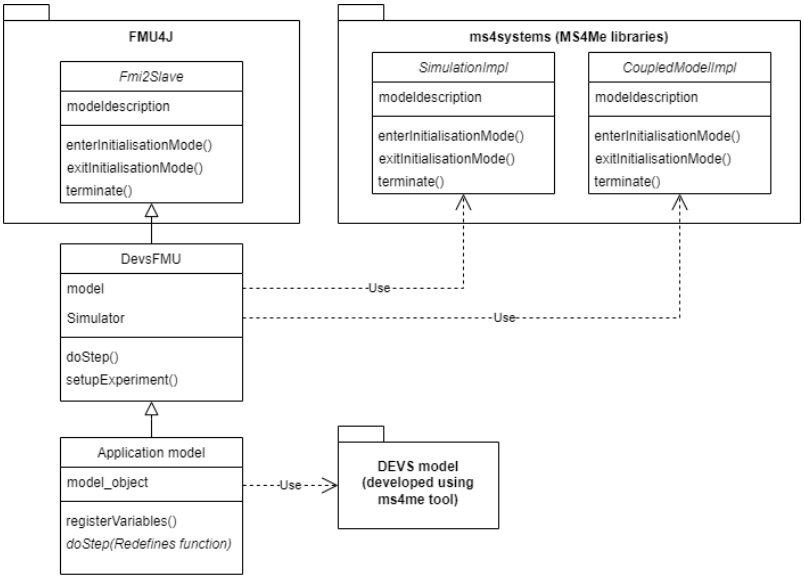


Figure 4: Design of DEVS model export.

### 3.3 Solvers

If a shared model is a continuous-time model, it needs to be simulated with the help of a continuous-time solver, like QSS. We defined first-order and second-order QSS solvers for state variable quantization and the numerical integration of the quantized variables. The QSS solver design in Cadmium determines the quantized state and calculates the time advance based on the quantum size and the derivative of the state variable. This is achieved by defining a function that takes the value of the state variable, its derivative, and the quantum size into account to calculate the value of the time advance (next step for integration). A constant quantum size is defined for discretizing state variables using uniform quantization. Hysteresis was also included in the design to prevent infinite events in a given fixed time (Joshi et al. 2024). A second-order QSS2 was designed as shown in (Kofman, 2002). The derivative value, quantum size, and variable value are supplied to the function, which calculates the value of sigma. The imported continuous time model shares the variables with the atomic model. The solvers can also be integrated with co-simulation operations to reduce the number of simulation instances. The co-simulation FMU includes a solver that performs numerical integration of the variables based on the specified time. This integration is performed only when the co-simulation algorithm tasks the FMU with doing so. The frequency of performing these integration operations can be limited using the quantizer. The atomic model co-simulating with the FMU reads the current value of the continuous-time variable. It calculates the derivative of the variable using its last value and time since the last read. It then calls the quantizer function to calculate the time for the next step (time advance), based on the specified quantum size and the function's derivative. The atomic model shares this value with the co-simulation FMU as the next communication point. After this calculated time, the model requests the FMU to complete the integration of values. It then re-reads the values of the continuous-time variables. The process is divided into two parts: Cadmium handles the quantization, while the FMU is responsible for performing the integration, with continuous-time variables being re-evaluated at each step.

## 4 MODELING THE STEERING CONTROL OF UGV

After discussing the framework used for the M&S, let us discuss the design of the steering control system for an Unmanned Ground Vehicle. Here, we considered a servo-controlled steering system that adjusts to the specified input angle. This steering system is connected to the four-wheeled base of the vehicle. For a mathematical model in this control operation, we referred to a paper (Haytham et al. 2014), where the author

conducted experiments to get the real-world values of coefficients and constants. The steering control system is controlled using a navigation system that sends values of velocity and angle. A PID controller is used to rotate the vehicle to the desired angle. The navigation system is developed using MS4Me, and the PID controller is designed and tuned using MATLAB (Simulink). We built a DEVS-based M&S of the Steering Control System in Cadmium, integrating the models developed in MS4Me and MATLAB through FMUs. The design makes use of the framework that we discussed earlier.

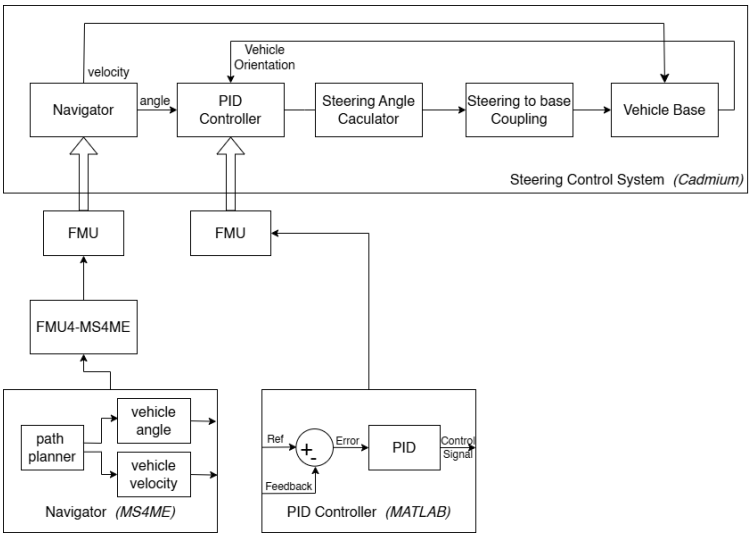


Figure 5: Steering Control System in Cadmium.

Figure 5 shows a block diagram with the design of the Steering Control system. We developed this with the help of three tools: Cadmium, MS4Me, and MATLAB. The Navigator system is created in MS4Me using DEVS formalism. This system consists of three atomic models. One determines the path, while the other two process the path data for transmission to the steering control system. The vehicle velocity block sends the value to the vehicle base, and the vehicle angle sends the value of the desired angle to the input of the PID controller. This model is then converted to a co-simulation FMU and subsequently shared with the Cadmium model, where the Navigator atomic model communicates with the FMU for M&S.

The continuous-time PID controller model is developed using MATLAB. This model has an error as its input and gives a control signal based on the error value. Figure 5 shows a PID controller model developed using the PID blocks, which receive the error signal from another block that calculates the error value using reference and feedback. This block in MATLAB is exported as a co-simulation FMU, as MATLAB inherently supports FMU export. This FMU is shared with the Cadmium model, where the PID controller atomic model communicates with the FMU for M&S.

The Cadmium block shows all the components developed. Each block represents an atomic model. The navigator model communicates with the FMU and sends the desired values of angle and velocity. The PID controller receives this angle value and uses it as a reference while it receives the actual robot orientation as feedback. The PID controller model uses these values to communicate with the FMU and obtain the control signal. This control signal is sent to the steering angle calculator, which calculates the new steering angle and sends it to the next block. The next model represents the connection between the base and the steering, and it calculates the angle of the wheels relative to the vehicle based on the input steering angle. These angles are sent to the vehicle base model, which calculates the vehicle's orientation using the wheel's angles and the velocity of the wheels provided by the navigator. The current orientation of the vehicle is sent as feedback to the PID controller, which uses it as feedback for deciding the next control action.

Now, let us have a look at the M&S setup and results. To simulate this model, we first simulated it with a fixed angle and velocity and then developed a list of commands using the Navigator model. Here, we will see examples of both steps. We will see two examples of fixed-angle simulation with a target angle of 5 degrees and a target angle of 25 degrees. In both scenarios, the initial angle is set to 0. The Navigator model

sets the target angle and velocity of the vehicle at the start of the simulation (t=0). The PID controller model starts the rotation upon receiving the target angle. The results of this simulation with Euler's method and QSS solver are shown in the figure below. Euler's method was implemented within the PID controller model using its standard formulation to evaluate the behavior under fixed-step integration.

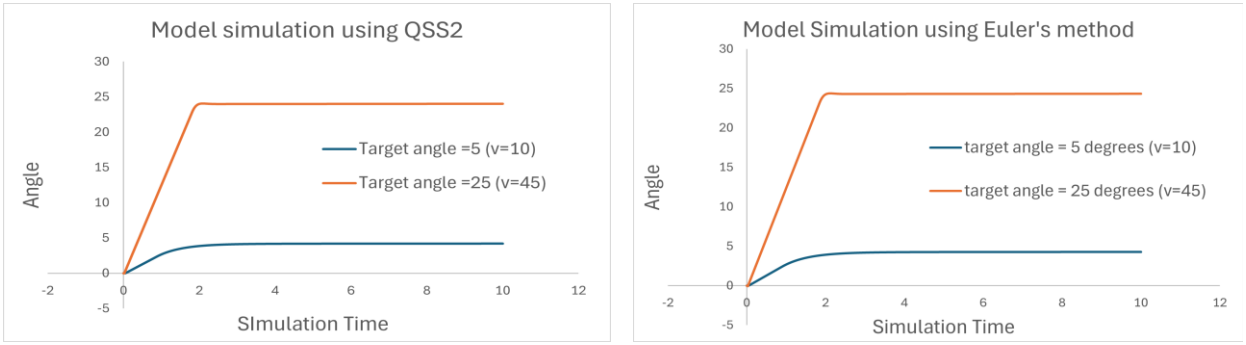


Figure 6: Simulation Results of fixed target angle.

Figure 6 shows the response of the vehicle orientation angle with the simulation time. The change of angle starts at time = 0 as soon as the target is set. The target angle 5 is simulated with velocity (v=10) and angle 25 is simulated with velocity (v=45). The controller reaches a steady state at approximately 2.5 s for the 5-degree angle and at 2 s for the 25-degree angle. The results of both the solvers are almost identical, with just a gap of 0.07 s to reach the steady state value.

In the second step, we developed a list of commands for the vehicle in the path planner of the Navigator model, and the model executes the steps as we discussed in the Navigator model description. The list of commands is listed in the table below.

Table 1: Commands for vehicle simulation

| Command/step | Angle (relative) | Distance | Output Angle (relative) | Output Velocity | Execution time |
|---|---|---|---|---|---|
| 1 | 25 | 400 | 25 | 40 | 10 |
| 2 | 55 | 600 | 80 | 40 | 15 |
| 3 | -40 | 400 | 40 | 40 | 10 |
| 4 | 50 | 500 | 90 | 40 | 12.5 |
| 5 | -10 | 500 | 80 | 40 | 12.5 |
| 6 | 40 | 250 | 120 | 40 | 6.25 |

In the table, the positive angle is a clockwise turn, and the negative value is an anticlockwise turn. These steps are designed in the Navigator model, and the output angle and velocity are sent to the PID controller atomic model and the vehicle base atomic model. The vehicle rotates and stays in that state until the next command is received. The steps are executed using Euler's method and QSS solvers. The value of the orientation angle of the vehicle through the simulation time is plotted in the figure below. This plot is generated using the simulation results stored in the .csv file, which are automatically generated by the Cadmium tool as simulation output.

Figures 7 shows the results of the steps listed in Table 1. The vehicle orientation plot shown in Figure 7-A is obtained from a simulation using the QSS solver with a quantum size of 0.001. For comparison, the same model was simulated using Euler's method, and the result is shown in Figure 7-B. Both simulations produced identical results. The initial value of vehicle orientation is 0. The first command is received at 10 s to achieve a 25-degree angle. The next command is sent after the calculated command execution time (10 s for command 1). At 20 s, the path planner sends the command to rotate 55 degrees, which is converted to an absolute value of 80 degrees and sent to the PID controller model. It then rotates the vehicle and stays in the same orientation until it receives the next command at 35 s to rotate 40 degrees anticlockwise.

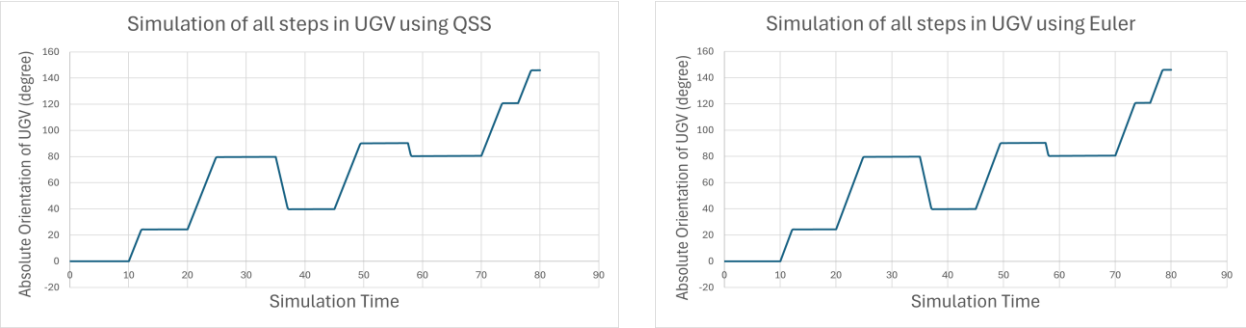


Figure 7: Simulation result showing vehicle orientation.

The vehicle orientation then changes to an absolute value of 40 degrees. At 45 s the next command to rotate 50 degrees clockwise sets the new target value as 90 degrees with a calculated execution time of 12.5 s. So, after 12.5 s the new command to rotate 10 degrees anticlockwise sets the target angle of 80 degrees. The vehicle stays at the orientation angle of 80 degrees until it receives the last command to rotate 40 degrees clockwise, creating an absolute angle of 120 degrees for a time of 6.25 s. After this command, the model repeats this set by doing the first command to rotate 25 degrees again.

## 5 CONCLUSION

Model sharing plays a vital role in the development of multifaceted systems by providing a means of information exchange. While reducing redundancy, it promotes collaborative problem-solving across industries. The Functional Mockup Interface (FMI) helps in collaboration by standardizing model sharing and making integration easier and more effective. Integrating this standard with M&S techniques like Discrete Event System Specification (DEVS) can further enhance collaborative efforts.

We introduced a framework for integrating the Functional Mockup Interface with the DEVS framework and used it to simulate a robotic application. To achieve this, we created an interface in Cadmium that follows the FMI standard for model import and integration. To simulate continuous-time models shared via FMU, we developed continuous-time solvers. We also designed an approach for exporting DEVS models as a co-simulation FMU. These exported models were integrated into Cadmium using the import interface. A time synchronization mechanism was established between the exported FMU and the model. The framework can communicate with multiple FMUs to perform integrated simulations.

For model simulation, the model sub-components were created using MATLAB and MS4Me. These were exported and integrated with the sub-components developed in Cadmium through the integration framework. This integration enables the advantages offered by DEVS-based simulation without restricting model development to DEVS-specific tools, which allows the use of domain-specific tools that best align with the requirements of the application. In future work, exported models will be shared with tools that use different formalisms to enhance interoperability and improve the flexibility of model sharing and reuse.

## REFERENCES

Astrom, Karl J. 1995. "PID controllers: theory, design, and tuning." *The international society of measurement and control.*

Blochwitz, Torsten, Martin Otter, Martin Arnold, Constanze Bausch, Christoph Clauß, Hilding Elmqvist, Andreas Junghanns, Jakob Mauss, Manuel Monteiro, and Thomas Neidhold. 2011. "The functional mockup interface for tool independent exchange of simulation models." *Proceedings of the 8th international Modelica conference.* Dresden: Linköping University Press.

n.d. *Cadmium V2: an object-oriented C++ M&S platform for the PDEVS formalism.* Accessed August 24, 2024. https://github.com/SimulationEverywhere/cadmium_v2.

Camus, Benjamin, Thomas Paris, Julien Vaubourg, Yannick Presse, Christine Bourjot, Laurent Ciarletta, and Vincent Chevrier. 2018. "Co-simulation of cyber-physical systems using a DEVS wrapping strategy in the MECSYCO middleware." *Simulation* 94 (12): 1099-1127.

Cárdenas, Román, and Gabriel Wainer. 2022. "Asymmetric Cell-DEVS models with the Cadmium simulator." *Simulation Modelling Practice and Theory* 121.

Cellier, François E, and Ernesto Kofman. 2006. *Continuous system simulation.* Springer Science & Business Media.

Chen, Dechao, Wang Zhixiong, Zhou Guanchen, and Li Shuai. 2022. "Path planning and energy efficiency of heterogeneous mobile robots using cuckoo–beetle swarm search algorithms with applications in UGV obstacle avoidance." *Sustainability 14* 15137.

Cherukuri, AKHIL, and Neti ADITYA SHASHANK. 2018. "Unmanned ground vehicle for military purpose." *International Journal of Pure and Applied Mathematics* 19 (12): 13189-13193.

Farella, Alessia, Francesco Paciolla, Tommaso Quartarella, and Simone Pascuzzi. 2024. "Agricultural unmanned ground vehicle (UGV): a brief overview." *International Symposium on Farm Machinery and Processes Management in Sustainable Agriculture.* Springer. 137-146.

n.d. *Functional Mockup Interface Standard.* Modelica Association. Accessed July 9, 2024. https://fmi-standard.org/assets/releases/FMI_for_ModelExchange_and_CoSimulation_v2.0.pdf.

Gomes, Cláudio, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. 2018. "Co-simulation: a survey." *ACM Computing Surveys (CSUR)* 51 (3): 1-33.

Hatledal, Lars Ivar, Houxiang Zhang, Arne Styve, and Geir Hovland. 2018. "Fmi4j: A software package for working with functional mock-up units on the java virtual machine." *The 59th Conference on Simulation and Modelling (SIMS 59).*

Haytham, A, Yehia Z Elhalwagy, Amr Wassal, and NM Darwish. 2014. "Modeling and simulation of four-wheel steering unmanned ground vehicles using a PID controller." *2014 international conference on engineering and technology (ICET).* IEEE.

Hebert, Martial H, Charles E Thorpe, and Anthony Stentz. 2012. *Intelligent unmanned ground vehicles: autonomous navigation research at Carnegie Mellon.* Vol. 388. Springer Science & Business Media.

Hu, Xi, and Rayan H Assaad. 2023. "The use of unmanned ground vehicles and unmanned aerial vehicles in the civil infrastructure sector: Applications, robotic platforms, sensors, and algorithms." *Expert Systems with Applications* (Elsevier) 120897.

Inzillo, Marco, and Kavka Carlos. 2017. "Multi-disciplinary Optimization with Standard Co-simulation Interfaces." *ICSOFT.* 453-458.

Joshi, Ritvik, James Nutaro, Bernard P Zeigler, Gabriel Wainer, and Kim Doohwan. 2024. "Functional Mock-up Interface Based Simulation of Continuous Time System in CADMIUM." *Annual Simulation Conference (ANNSIM'24).* American University, DC, USA.

Kim, Sungung, Hessam S Sarjoughian, and Vignesh Elamvazhuthi. 2009. "DEVS-suite: a simulator supporting visual experimentation design and behavior monitoring." *SpringSim* 9: 1-7.

Kofman, Ernesto. 2002. "A second-order approximation for DEVS simulation of continuous systems." *Simulation* 78 (2): 76-89.

Kofman, Ernesto, and Sergio Junco. 2001. "Quantized-state systems: a DEVS Approach for continuous system simulation." *Transactions of The Society for Modeling and Simulation International* 18 (3): 123-132.

Lin, Xuanli. 2021. *Co-simulation of Cyber-Physical Systems Using DEVS and Functional Mockup Units.* Arizona State University.

Lindholm, Tim, Frank Yellin, Gilad Bracha, and Alex Buckley. 2013. *The Java virtual machine specification.* Addison-wesley.

Müller, Wolfgang, and Edmund Widl. 2013. "Linking FMI-based components with discrete event systems." *2013 IEEE International Systems Conference (SysCon).* Orlando, FL: IEEE. 676-680.

Ragothaman, Sonya, Maaref Mahdi, and Kassas Zaher M. 2021. "Autonomous ground vehicle path planning in urban environments using GNSS and cellular signals reliability maps: Models and algorithms." *IEEE Transactions on Aerospace and Electronic Systems.* 1562-1580.

Seo, Chungman, Bernard P Zeigler, Robert Coop, and Doohwan Kim. 2013. "DEVS modeling and simulation methodology with MS4 Me software tool." *SpringSim (TMS-DEVS)* 33.

Tripakis, Stavros. 2015. "Bridging the semantic gap between heterogeneous modeling formalisms and FMI." *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS).* IEEE. 60-69.

Vanommeslaeghe, Yon, Bert Van Acker, Joachim Denil, and De Meulenaere Paul. 2020. "A co-simulation approach for the evaluation of multi-core embedded platforms in cyber-physical systems." *Proceedings of the 2020 Summer Simulation Conference.* ACM. 1-12.

—. 2024. "Integrating DEVS and FMI 3.0 for the Simulated Deployment of Embedded Applications." *2024 Annual Modeling and Simulation Conference (ANNSIM).* Washington DC: IEEE. 1-13.

Wang, Liuping. 2020. *PID control system design and automatic tuning using MATLAB/Simulink.* John Wiley & Sons.

Zeigler, Bernard P, and Jong Sik Lee. 1998. "Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment." *Enabling Technology for Simulation Science II.* SPIE. 49-58.

Zeigler, Bernard P, Herbert Praehofer, and Tag Gon Kim. 2000. *Theory of Modeling and Simulation.* Academic press.

Zhang, Heming, Wang Hongwei, Chen David, and Zacharewicz Gregory. 2010. "A model-driven approach to multidisciplinary collaborative simulation for virtual product development." *Advanced Engineering Informatics 24.* 167-179.

## AUTHOR BIOGRAPHIES

**RITVIK JOSHI** is a Systems Software Developer at BlackBerry QNX. He completed his M.A.Sc. at the Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada. His research interests include robotics, embedded systems, real-time systems, modeling, and simulation. His email address is ritvikjoshi@cmail.carleton.ca.

**JAMES NUTARO** is Group Lead for the Computational Systems Engineering & Cybernetics Group at Oak Ridge National Laboratory. He holds a Ph.D. in Computer Engineering from the University of Arizona. His research interests discrete event systems, systems modeling and simulation, and hybrid dynamic systems. His email address is nutarojj@ornl.gov.

**BERNARD ZEIGLER** is Professor Emeritus of Electrical and Computer Engineering at the University of Arizona (USA) and Chief Scientist of RTSync Corp. (USA). Dr. Zeigler is a Fellow of IEEE and SCS and received the INFORMS Lifetime Achievement Award. He is a co-director of the Arizona Center of Integrative Modeling and Simulation. His email address is zeigler@rtsync.com.

**GABRIEL A. WAINER** is Professor in the Department of Systems and Computer Engineering at Carleton University (Ottawa, ON, Canada). He is the head of the Advanced Real-Time Simulation lab, located at Carleton University's Centre for Advanced Simulation and Visualization (V-Sim). He is an ACM Distinguished Speaker and a Fellow of SCS. His email address is gwainer@sce.carleton.ca.

**DOOHWAN KIM** is the founder and president of RTSync Corp., which specializes in Predictive Analytics and Model-Based System Engineering based on DEVS M&S. He received his Ph.D. degree from the University of Arizona in 1996. His email address is dhkim@rtsync.com.