Model-Based Integration Platform for FMI Co-Simulation and Heterogeneous Simulations of Cyber-Physical Systems

Himanshu Neema[1] Jesse Gohl[2] Zsolt Lattmann[1]
Janos Sztipanovits[1] Gabor Karsai[1] Sandeep Neema[1]
Ted Bapty[1] John Batteh[2] Hubertus Tummescheit[2]
[1]Institute for Software Integrated Systems, Vanderbilt University
1025 16th Avenue South, Suite 102, Nashville, TN 37212, USA
[2]Modelon, Inc.
2389 Main Street, Glastonbury, CT 06033, USA
himanshu@isis.vanderbilt.edu jesse.gohl@modelon.com lattmann@isis.vanderbilt.edu
sztipaj@isis.vanderbilt.edu gabor@isis.vanderbilt.edu sandeep@isis.vanderbilt.edu
bapty@isis.vanderbilt.edu john.batteh@modelon.com hubertus.tummescheit@modelon.com

Abstract

Virtual evaluation of complex Cyber-Physical Systems (CPS) with a number of tightly integrated domains such as physical, mechanical, electrical, thermal, cyber, etc. demand the use of heterogeneous simulation environments. Our previous effort with C2 Wind Tunnel (C2WT) attempted to solve the challenges of evaluating these complex systems as-a-whole, by integrating multiple simulation platforms with varying semantics and integrating and managing different simulation models and their interactions. Recently, a great interest has developed to use Functional Mockup Interface (FMI) for a variety of dynamics simulation packages, particularly in Commercial Off-The-Shelf (COTS) tools. Leveraging the C2WT effort on effective integration of different simulation engines with different Models of Computation (MoCs), we propose, in this paper, to use the proven methods of High-Level Architecture (HLA)-based model and system integration. We identify the challenges of integrating Functional Mockup Unit for Co-Simulation (FMU-CS) in general and via HLA and present a novel model-based approach to rapidly synthesize an effective integration. The approach presented provides a unique opportunity to integrate readily available FMU-CS components with various specialized simulation packages to rapidly synthesize HLA-based integrated simulations for the overall composed Cyber-Physical Systems.

Keywords: Functional Mockup Interface, Functional Mock-up Unit for Co-Simulation, Cyber-Physical Systems, Heterogeneous simulation, Multi-paradigm modeling, Model-based integration, DSML, Distributed Simulation, High-Level Architecture

1 Introduction

Cyber-Physical Systems (CPS) [1] are composed of several collaborating physical and computing components that interact through embedded communication capabilities. These systems require advanced integration of abstractions and techniques that have been developed over the past years in disparate areas such as cyber systems that rely heavily on computation and networking and physical systems that employ various engineering methods in domains such as mechanical, thermal, electrical, electronic, hydraulic, thermal, biological, and acoustic.

Analysis of Cyber-Physical Systems poses unique challenges due to the heterogeneity of components and interactions [2]. The fundamental differences in the characteristics of these different physical and computation processes lead to a huge spectrum of modeling methods. For example, some components can be easily described by differential equations, while others like communication networks typically require Discrete-Event Simulation (DEVS) techniques. As such, several simulation tools and techniques are needed for CPS simulation and analysis. This further necessitates an over-arching CPS model and system integration platform that is model-based and supports rapid synthesis of distributed heterogeneous CPS simulations.

Co-Simulation (Co-operative Simulation) is a simulation method that permits simulating individual components using different simulation tools simultaneously and collaboratively. Individual simulation tools exchange information such as individual system variables and their values, time steps for synchronization, and control signals for orchestrating the co-operative simulation. In this way, engineers can use different simulation tools together to create virtual prototypes of entire Cyber-Physical Systems. In practice, however, significant challenges remain with regard to the syntax and semantics of model and system integration.

In the Co-Simulation domain, a recent effort by the MODELISAR ITEA2 project that develops a tool independent standard called the Functional Mock-up Interface (FMI) [3] [4] [5] has gained significant influence, more prominently in the automotive industry. The FMI standard provides a well-defined set of function calls to specify simulation components. FMI-compliant simulations pack shared libraries that can be executed using the standardized function calls and the model execution must adhere to the rules of the standard. These function calls span all stages of the model execution, viz. initialization, configuration, access, modification, and manipulation.

The strength of FMI lies in the fact that all simulation tools participating in the Co-Simulation follow the defined standard and as such provides for standardized access to model equations. This permits coupling of Continuous-Time and Discrete-Time systems that are part and parcel of Cyber-Physical Systems. In some ways, this is also a limitation because not all simulation tools are amenable to support all of the strictly specified FMI function calls.

Another key requirement for Co-Simulation via FMI is to also develop a master algorithm that orchestrates the steps of Co-Simulation. Master algorithms must control the data exchange between subsystems and synchronize their individual simulations according to the requirements of the integrated simulation of the overall Cyber-Physical System. Although the FMI standard does not describe or limit the implementation of the master algorithm, the algorithm requirements and features often limit its implementation as a centralized orchestrator that can communicate effectively with all participating subsystems. Centralized nature not only can become a performance bottleneck, it can also serve as a single point of failure in the distributed simulation's computational infrastructure.

Furthermore, as Cyber-Physical Systems involve vastly different sub-domains and physical processes that vary greatly in the execution frequency at which they need to run. This leads to significantly different dynamic response characteristics in terms of frequencies. For example, mechanical components of a complex CPS often have much slow frequency responses compared to fast electronic components. Single standalone monolithic model of a CPS therefore suffers heavily with solver inefficiencies. These systems are generally highly complex and have a significant non-linearity and discontinuities, which further adds to inefficiencies of solvers. Taking subsystems apart and using different solver step-sizes offers a potential solution. However, multirate composition also introduces some inefficiencies due to clock management, composition restrictions, data exchange, and potential stability issues if the system is split at the wrong place.

Another effort developed by U.S. Modeling and Simulation Coordination Office (M&S CO) is the High Level Architecture (HLA) [6]. The HLA provides a specification of a common technical architecture for modeling and simulation with a primary goal to facilitate interoperability among simulations and to promote re-use of simulations and their components. The HLA comprises of three major components: HLA rules, HLA interface specification, and HLA object model template [6]. With these rules, the HLA standardizes run-time support for various tasks, such as coordinated time evolution, message passing and shared object management. The key difference from FMI is that HLA regards individual simulation components at the level of processes as opposed to libraries. This enables broader integration of different simulation tools with different Models of Computation. Even Functional Mock-up Units can be integrated as a participating simulation tool in the overall integrated simulation of the Cyber-Physical Systems.

Another key benefit of HLA is that its Distributed Discrete Event model of computation allows full flexibility to individual subsystems in using any internal solver and model of computation. Moreover, this flexibility permits multirate simulations by design.

However, the HLA standard also lacks some key facilities for developing integrated distributed heterogeneous simulations. For example, the HLA standard does not formalize methods for developing interactions and objects used by HLA federates and it does not provide facilities for easily moving simulations from one computational node to other. Consequently, HLA-based simulations also require a significant amount of tedious and error-prone hand-developed integration code.

Achieving the integrated simulation of Cyber-Physical Systems require effective integration of a huge spectrum of models of physical processes, communication systems, exchanged information, and control mechanisms. As detailed above, the approaches of FMI and HLA both have their advantages and some key limitations. The approach of using HLA as a master algorithm enables use of FMUs in a Co-Simulation environment while also providing flexibility of using other types of non-FMU simulations [9]. The resulting framework can provide a much broader scale of simulation tools that can be used in the integrated simulation of Cyber-Physical Systems. However, several gaps need to be filled in order to develop a platform that enables this integration in an efficient manner. A single efficient model-based platform is needed that:

- Enables modeling of interactions and shared objects between simulation tools
- Enables modeling of integration of systems with their data exchange mechanisms
- Enables modeling of deployment of simulation tools on computational infrastructure
- Enables a "decentralized" master algorithm for FMI Co-Simulation
- Enables multirate modeling with dynamic management of subsystem clock rates
- Provides a set of tools to generate necessary artifacts for rapid synthesis of simulations

This paper attempts to address these important challenges in creating a single coherent platform for developing integrated distributed simulations of Cyber-Physical Systems. We build upon our previous work on a model-based integration platform called the Command and Control Wind Tunnel (C2WT) [7] [8].

The rest of the paper is organized as follows. Section 2 and 3 give an overview of the C2 Wind Tunnel and FMI for Co-Simulation respectively. We present our detailed model-based integration approach in Section 4 and provide a detailed case study with experimental results in Section 5. Finally, Section 6 concludes the paper.

2 C2 Wind Tunnel

Over the past several years, we have developed a model-based multi-model integration platform called the Command and Control Wind Tunnel (C2WT) [7] [8]. It is an integrated, graphical, multi-model, distributed simulation environment for the experimental evaluation of large-scale C2 systems with various organizational and technical architectures. It enables a variety of simulation engines to interact and transmit data from one another and log and analyze simulation results. Figure 1 below gives a conceptual architecture of C2WT.

The High-Level Architecture is a standardized framework for distributed computer simulation systems. Communications between different federates is managed via the Run-Time Infrastructure (RTI) layer. The RTI provides a set of services such as time management, data distribution, message passing, and ownership management. Other components of the HLA standard are the Object Model Template (OMT) and the Federate Interface Specification (FIS).

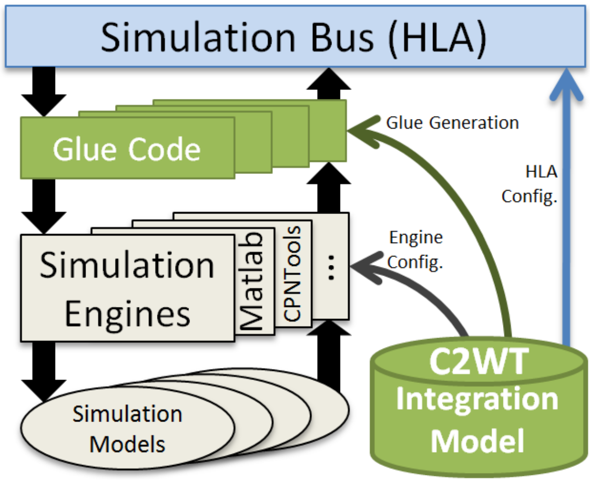


Figure 1: Conceptual architecture of C2WT

The HLA standard focuses on three primary areas. First is time coordination throughout the federation. The evolution of time is a key thread through each of the integrated simulators. Each simulation platform must slave its progression of time to that of the overall HLA clock. The HLA standard provides several methods by which to accomplish this. Second is coordination of inter-federate messages and shared data objects. The HLA standard provides a publish-and-subscribe mechanism for passing messages and object updates throughout the federation. Third, the HLA standard provides for basic simulation execution control. Starting, pausing, and stopping the execution of a simulation is built directly into the HLA standard. The C2 Wind Tunnel relies upon all of these services during run-time.

As HLA is an accepted standard, a number of commercial, academic, and alternate RTI implementations are available. Currently, we use the Portico RTI [10] – which provides support for both C++ and Java clients and is compliant with version 1.3 of the HLA standard.

The HLA provides a standard for the RTI that supports the coordinated execution of distributed simulations. However, designing the model integration, coding the platform-to-RTI glue-code, and testing and deploying all of the various run-time components across multiple platform-specific simulation tools is a highly challenging task. C2WT provides a solution to this simulation integration problem. It provides a holistic modeling and management environment built around a custom Domain-Specific Modeling Language (DSML) [11], implemented in Generic Modeling Environment (GME) [11], and a related suite of model interpreters to coordinate between the integration model and the platform-specific simulation tools involved in the overall environment. It facilitates the rapid development of integration models and use of these models throughout the lifecycle of the simulated environment. With simulation engine specific model configurations and experiment specific deployment modeling, it enables significant automation in the development of integrated distributed simulation. With integration modeling support and various sophisticated generation tools, C2WT provides a robust platform for users to rapidly model and synthesize complex, heterogeneous, command and control simulations.

3 FMI for Co-Simulation

Functional Mock-up Interface (FMI) [3] [4] [5] was initiated and organized by Daimler AG within the ITEA2 project MODELISAR [3]. The FMI standard consists of two main parts. The first part is FMI for Model Exchange, which standardizes the distribution of a dynamic system model in the form of generated C-Code as an input/output block to other simulation environments. The second part is FMI for Co-Simulation, which standardizes the mechanisms for coupling of two or more simulation tools in a co-simulation environment.

The key idea is to have a discrete set of communication points only, at which times the subsystems exchange any data. Outside of these points, the subsystems are executed independently. The data exchange is controlled by a master system that also manages time synchronization of subsystems.

The FMI Co-simulation master simulator couples the subsystem simulators through a zip-archive. This zip-archive contains shared library files (.DLL, .SO) that conform to the function call specifications given in the standard. Each zip-archive also contains a XML file that provides meta-data and further details of the model such as default start and stop times, variable types, units, tool specific data, parameter and variable names and attributes. The XML also contains specification for executing the model as a shared library during a simulation run (CoSimulation_Standalone) or by importing a slave tool wrapper and interfacing it with the external tool (CoSimulation_Tool).

4 Model-Based Integration

One of the primary contributions of our effort is our focus on developing a completely model-based integration approach. Our efforts leverage the Generic Modeling Environment (GME) [11] tool suite for designing the integration model DSML [11] and HLA [6] to provide run-time support as the "simulation bus".

4.1 Needs and Challenges

Cyber-Physical Systems [1] [2] are highly complex and their simulation spans a multitude of computational domains and specializations. A large number of tools exist that have been developed for specific aspects of CPSs. A variety of tools exist even for a single aspect of CPSs. For example, many special purpose simulation tools exist to model and analyze vehicle dynamics or for switching mechanisms of hybrid drivetrains. As such the integration platform must be open toward use of any tool that may be required for some component/aspect of the CPS simulation.

A subtle problem in using multiple simulation tools in an integrated simulation is that they tend to use many different Models of Computation (MoC). For example, Discrete-Event, Discrete-Time, Continuous-Time, Synchronous Dataflow, are among the many MoCs used. Each MoC has a specific mechanism for time progression and event handling. The integration platform must be able to handle tools that use different MoCs in highly flexible manner. The integrated system must respect time synchronization with other simulation tools as well as the causality of events must be preserved. In addition to system integration, the platform must also enable integration of models by means of capturing the communication (with any translation that might be needed) that occurs between them.

As a general rule, it is preferable to have a graphical environment that provides well-defined semantics for modeling concepts, their relations, and rules for composition. Moreover, for rapid synthesis of simulations, the platform must support tools for translation of models to executable software that conform to specified executable semantics. The automation not only provides efficient development of simulations, it also significantly minimizes human errors.

The integration environment should also provide capabilities for modeling and configuration experimentation and logging.

Furthermore, when FMUs are integrated the rules of FMI must still be adhered to. Particularly, the models in the FMUs must be accessed, controlled, and manipulated using the function calls specified in the FMI standard.

4.2 Meta-modeling

The Generic Modeling Environment is a meta-programmable model-integrated computing (MIC) [11] toolkit that supports the creation of rich domain-specific modeling and program synthesis environments. Configuration is accomplished through meta models, expressed as UML class diagrams, specifying the modeling paradigm of the application domain. Meta models characterize the abstract syntax of the domain-specific modeling language, defining which objects (i.e. boxes, connections, and attributes) are permissible in the language. Another way to envision this is that a DSML [11] is a schema or data model for all the possible models that can be expressed by a language. Using finite state machines as an example, the DSML would consist of states and transitions. From these elements any state machine can be realized. The inherent flexibility and extensibility of the GME [11] via meta models make it an ideal foundation for the C2 Wind Tunnel environment. Alternate meta modeling frameworks have also been developed in the past, such as AToM3 [12], MetaCase [13], Microsoft DSL [14], and the Eclipse Modeling Framework [15].

4.3 Model-Based Integration of FMUs in C2WT

As detailed in section 2, C2WT provides an overarching modeling and management environment and a suite of model interpreters to coordinate the integration models and platform-specific simulation tools involved in the overall heterogeneous distributed simulations. The user is referred to [7] for details of the meta-modeling language and its executable semantics. In this section, we further discuss the integration of FMUs as HLA-federates in the C2WT platform.

In this work, the C2WT metamodel was further customized to enable FMU specific federate specifications. Although the original C2WT metamodel is sufficient to support integration of newer types of federates, having simulation tool/technique specific first-class objects in the modeling language makes reasoning about such entities more flexible and can support extensive automation. The FMU-federate model specifies the location of the zip archive, whether to log variable values during simulation, additional variables (other than input and output) to log, and ratio of macro and micro steps for multirate simulations.

Figure 2 below shows the extension to the original C2WT architecture to incorporate FMU federates in the platform.

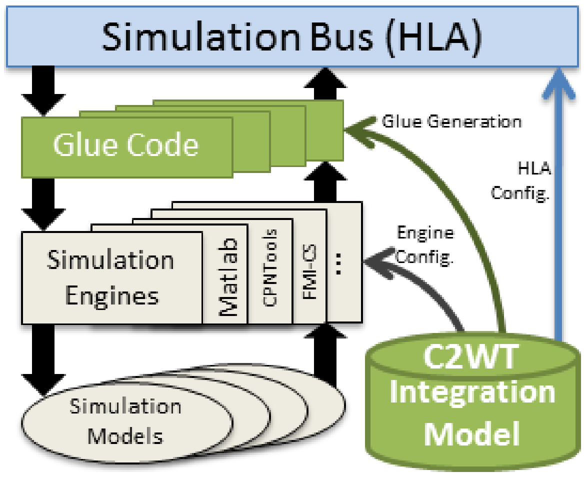


Figure 2: C2WT extended for FMI-CS

Our model interpreters can read the models with specified input and output relationships with other simulation tools and even other FMUs and can automatically generate all the executable code that can be deployed on different nodes in the available computational infrastructure for the simulation. As previously mentioned, C2WT supports simple modeling of computational infrastructure and assignment of federates on its nodes.

Following the rules of FMU access, modification, and manipulation as described in the FMI standard [3] [4], we developed a simplified procedure for FMU-federate execution as given below:

Initialization phase (before simulation start):

```
1: Load FMU zip archive, read model description
2: Load shared libraries in the FMU
3: Instantiate the FMU slave
4: Setup input/output and HLA-interaction maps
5: Setup up logging
```

Execution phase (during simulation):

```
1: Synchronize start of simulation with all tools
2: Request RTI to proceed to step-size and wait
3: Update input variables with HLA updates
4: Call doStep in step-size/#micro-steps chunks
5: Continue #4 until full step-size is executed
6: Update HLA with output variables
7: Go to #2
```

Please note that above is rather simplified procedure of FMU integration mechanism in C2WT. The actual implementation also involves setting up statistical and database logging, micro-step management to avoid overlaps, error-handling, efficient federate code execution, reliable & reusable time advancing facilities, and model state and HLA interaction synchronization.

5 Case Study

To illustrate our model-based approach for FMU integration in C2WT we present a high-fidelity model of a representation of a Vehicle Thermal Management (VTM) system which is intended for studying interactions of thermal management systems within a vehicle.

5.1 Model description

This particular example is a conventional four wheel chassis and drivetrain architecture with a spark ignition engine and standard transmission. These mechanical systems are created using components from the Vehicle Dynamics Library (VDL) from Modelon [16]. The model also includes a representation of the coolant loop for the engine and transmission oil loop in conjunction with a four heat exchanger stack for the thermal domain. These portions of the model are constructed from components of the Liquid Cooling Library (LCL) from Modelon. A snapshot of the overall model is shown in the following Figure 3 below.

The key component models of the system are: Driver, Vehicle (Engine, Transmission, Driveline, Chassis, Aerodynamics, External loads, and Brakes), Lumped engine thermal mass, Lumped transmission thermal mass, Engine coolant fluid circuit, Transmission oil cooling circuit, Heat exchanger stack, Low voltage battery, Alternator, Cooling fan and controller, and Grill shutters and controller. Table 1 below provides key features of these component models.

Since the purpose of this model is to study vehicle thermal dynamics, a simplified 1D longitudinal dynamics chassis model is used rather than a full 3D body model. This allows for faster simulations of the typically long duration drive cycles.

During the simulation, heat that is generated by the engine is stored within the engine thermal mass and then rejected to the coolant-to-air heat exchanger (radiator) through a coolant fluid loop. A similar loop and heat exchanger also exists for the transmission.

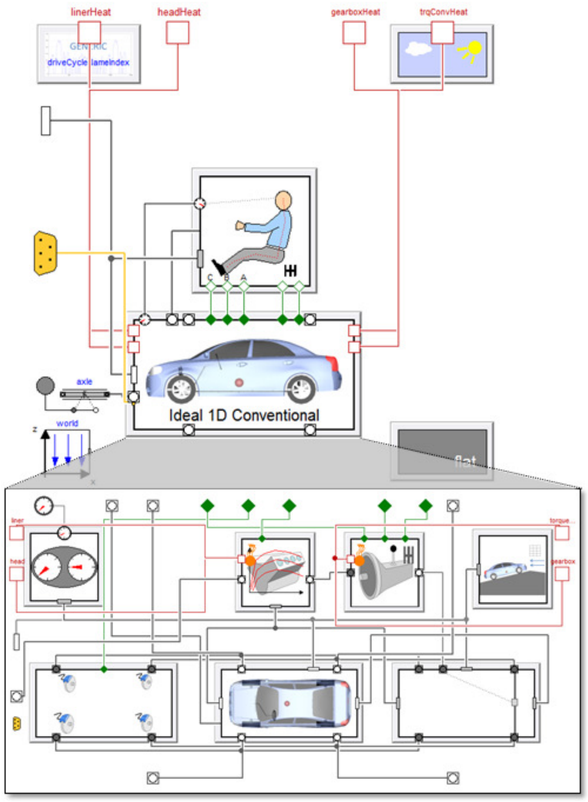


Figure 3: Overall system model

Component model	Key features
Driver	• Closed loop speed control for drive cycle following with auto gear shifting • Standard list of drive cycles
Vehicle - Engine	• Torque map from throttle and speed with heat generation
Vehicle – Transmission	• Standard transmission with efficiency losses for heat generation
Vehicle – Driveline	• Rear wheel drive with parameterized final drive ratio
Vehicle – Chassis	• VDL compatible interfaces • 1D longitudinal dynamics • Ideal suspension and wheels
Fluid coolant circuits	• Heat absorption from thermal masses • Crankshaft pump loads • Thermostat fluid control • Fluid flow resistances
Heat exchanger stack	• Parameterized geometry, efficiency, resistances • Stack ordering effects • Air flow effects due to vehicle speed, grill position and fan speed
Alternator	• Crankshaft load
Cooling fan and controller	• Power supplied by alternator
Grill shutters and controller	• Variable position • Heat exchanger stack air flow modification • Aerodynamic drag effects

Table 1: Key features of component models

The model is well suited to thermal management controller design, studying tradeoffs between thermal management energy demands and fuel economy, heat exchanger efficiency and sizing, and coolant fluid flow dynamics.

For this paper, the model was partitioned into separate executables by dividing the model along domain boundaries. In this case the vehicle mechanics, electrical, and driver were grouped into one model while the fluid and thermal portions of the model were grouped into another. This partitioning allows for execution of Driver vehicle and Thermal management parts at different rates. Owing to the inclusion of fluid portions in the Thermal management part, this part needed to run with a much lower step-size than the Driver vehicle part to maintain system stability.

In order to do this the physical connections that are bisected by the boundaries must be converted to causal signals. As an example for the engine, the heat is generated within the mechanical portion of the model. The heat is directed to the lumped thermal model, within the thermal portion of the model, which determines the thermal mass temperature. Images of these two systems are shown in Figures 4 and 5 below.

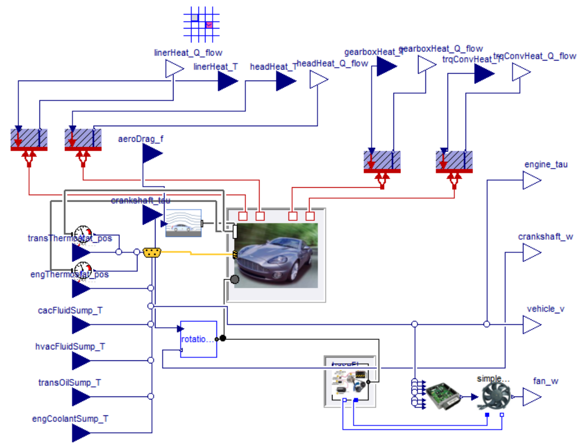


Figure 4: Driver vehicle model

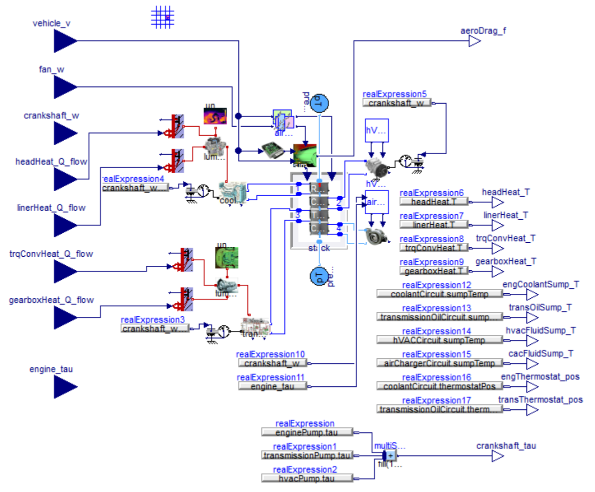


Figure 5: Thermal management model

5.2 Simulation architecture

The simulation setup consisted of mainly three federates, viz. Driver vehicle, Thermal management, and the Manager federate. Manager federate is an auto-generated external federate, which is used mainly as a front-end controller of the overall heterogeneous simulation. The simulation architecture is illustrated in the Figure 6 below.

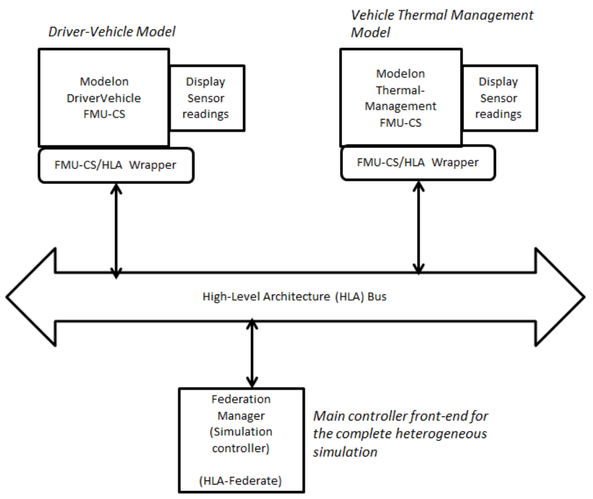

Driver-Vehicle Model
Modelon DriverVehicle FMU-CS
Display Sensor readings
FMU-CS/HLA Wrapper
Vehicle Thermal Management Model
Modelon Thermal-Management FMU-CS
Display Sensor readings
FMU-CS/HLA Wrapper
High-Level Architecture (HLA) Bus
Federation Manager (Simulation controller)
(HLA-Federate)
Main controller front-end for the complete heterogeneous simulation

Figure 6: Simulation architecture

5.3 Data and Integration model

The actual data and integration model are given in the Figures 7 and 8 below. These show the input and variables from the Driver vehicle and Thermal management federates. These two models are executed as FMUs in the C2WT.

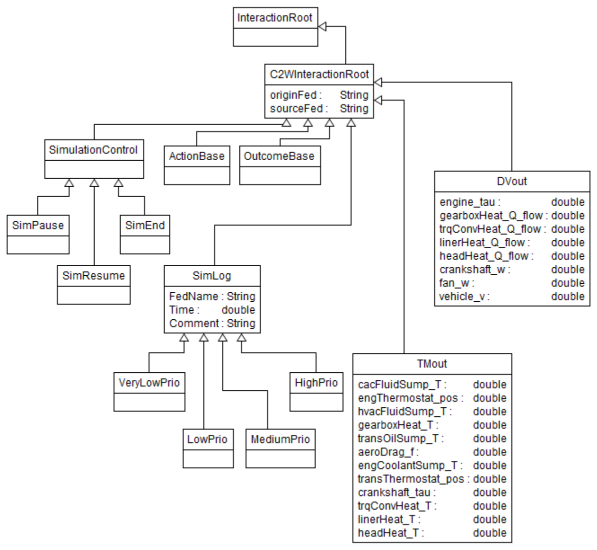


Figure 7: Data model

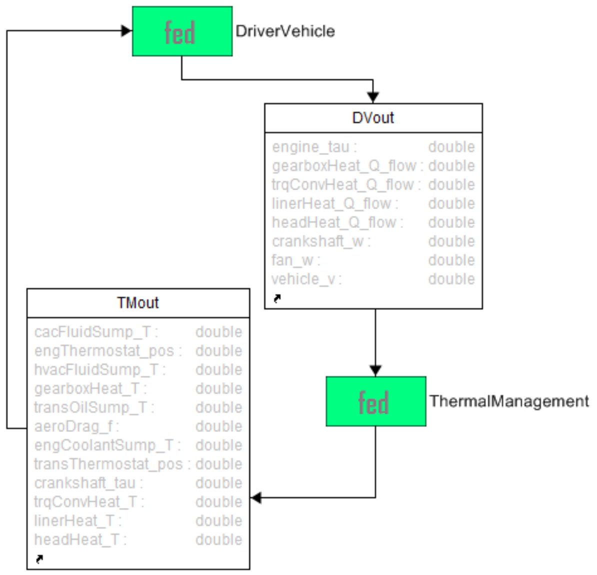


Figure 8: Integration model

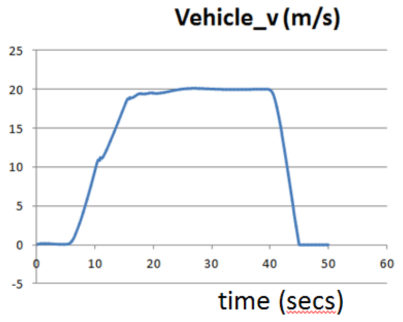
5.4 Experimental Results

For the experiment, the Driver vehicle and Thermal management FMUs were exported from Dymola [16] models by Modelon, Inc. [16]. We used a JFMI Ptolemy APIs [17] to connect the FMUs to our Java based C2WT platform. All federates were running in a single Ubuntu 32 virtual machine. The Run-Time Infrastructure (RTI) used was Portico [10]. Total simulation time for the experiment was 50 seconds.

The simulation was setup as a multirate simulation with different step-sizes for the three federates: Driver vehicle (10 ms), Thermal management (5 ms), and Federation Manager (100 ms). The entire simulation ran in about ~9 minutes. The Figures 9 and 10 above show the experimental results for the total 50 seconds of simulation time. It should be noted though that the VTM models used were currently not optimized for efficiency.

From the experimental results, we found closely matching plots with same peak and trough values that were in the equivalent single monolithic (combined Driver vehicle and Thermal management) model. The overall runtime (~9 minutes) was also comparable to standalone single model simulation time in Dymola (~6 minutes) despite the use of a third federate (viz. Manager federate) in the simulation and delays due to inter-process communications.

The models were developed with a variable step solver as requirement. However, they could still run with a fixed step solver (with a maximum step-size of 1.5 ms). However, with our setup of separating the Driver vehicle and Thermal management components as separate FMUs and executing them through C2WT platform, we could even execute these components at 10 ms and 5 ms step-sizes respectively.



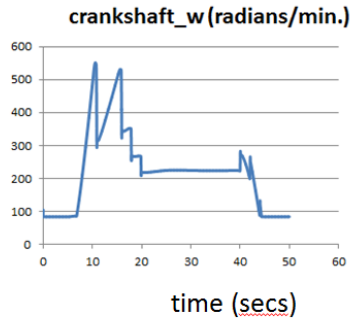
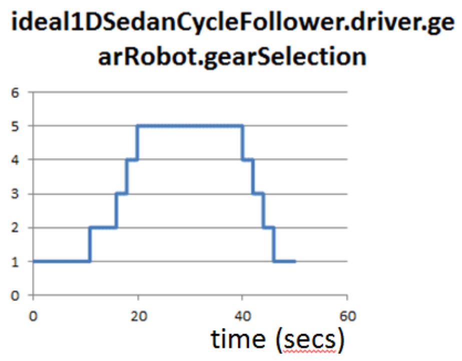


Figure 9: Vehicle speed and crankshaft angular velocity



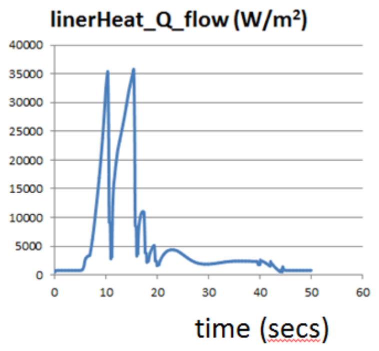


Figure 10: Gear selection and Liner heat flux

Yet another experiment we have performed is the one where we placed a network simulator for the CAN bus that must be placed between the above two components. We used the OMNeT++ simulator [18] to model that. In this experiment, we varied the rates of the FMUs to initially match the rate at which network simulator was run, viz. 0.5 ms, and then in the second setup we increased the step-size of Driver vehicle and Thermal management to 1 ms. We found that the results still matched while in the second setup they executed in about one-third the overall wall-clock time. We omit here further details of experiment setup for brevity.

6 Conclusions

In this paper, we have successfully demonstrated a model-based integration approach to rapidly synthesize multi-model distributed simulation that may also involve co-simulation FMUs as component models. The FMUs are automatically wrapped as HLA-federates that can be executed in the C2WT platform.

We also illustrated that different federates can be run with different clocks and their synchronization in C2WT is managed using HLA time management facilities. We have also integrated FMU-CS in simulations that also use other simulation tools such as a network simulator or a 3D terrain simulator. The integration of other federates in C2WT has been previously demonstrated in [7]. Thus C2WT provides a broader range of simulation tool integration that involves FMI and non-FMI simulations to enable development of System-of-System (SOS) simulations.

C2WT supports real-time and as-fast-as-possible modes of simulation execution. However, currently the real-time simulation requires that the individual component simulations can run faster than real-time.

C2WT also supports human-in-the-loop simulations with real-time simulations. In this case human interaction with running simulations (e.g. in military training exercises) is performed using HLA-interaction mappings.

One of the key benefits of C2WT platform is its support for extensive experimentation, message logging, state variables logging, and analysis support.

The research at our institute is currently ongoing with the applications of FMI Co-Simulation using HLA-based integrations. We anticipate novel methods for FMI Co-Simulations that are rapidly synthesized and may perform faster than single monolithic simulations.

We are also working on extending the C2WT platform to support other simulation techniques and tools such as SystemC.

7 Acknowledgements

The authors acknowledge financial support from the US DoD's Defense Advanced Research Projects Agency under the project "Adaptive Vehicle Make" [19]. We also acknowledge the invaluable contributions for the efforts in this paper from our collaborators at Modelon, Inc. [16].

References

[1] J. Sztipanovits, "Composition of cyber-physical systems," in Proc. of the 14th Annual IEEE Int'l. Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '07). Washington, DC, USA: IEEE Computer Society, 2007, pp. 3–6.

[2] E. Lee, "Cyber physical systems: Design challenges," in Proc. of the 11th IEEE Int'l. Symposium on Object Oriented Real-Time Distributed Computing (ISORC'08), May 2008, pp. 363–369.

[3] Functional Mock-up Interface – www.fmi-standard.org

[4] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H.Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J. V. Peetz, S. Wolf, "The Functional Mockup Interface for Tool independent Exchange of Simulation Models", in 8th International Modelica Conference, Dresden, 2011, pp. 20-22.

[5] Modelica Association: Modelica – A Unified Object-Oriented Language for Physical Systems Modeling. Language Specification, Version 3.2, March 24, 2010: www.modelica.org/documentas/ModelicaSpec32.pdf

[6] HLA standard – IEEE standard for modeling and simulation (M&S) high-level architecture (HLA) – framework and rules ieeexplore.ieee.org/servlet/opac?punumber=7179.

[7] Graham Hemingway, Himanshu Neema, Harmon Nine, Janos Sztipanovits, Gabor Karsai, "Rapid synthesis of high-level architecture-based heterogeneous simulation: a

model-based integration approach", Simulation 88(2), 217-232 (2012)

[8] C2WT community wiki – wiki.isis.vanderbilt.edu/OpenC2WT

[9] Awais, M.U.; Palensky, P.; Elsheikh, A.; Widl, E.; Matthias, S., "The high level architecture RTI as a master to the functional mock-up interface components," *Computing, Networking and Communications (ICNC), 2013 International Conference on* , vol., no., pp.315,320, 28-31 Jan. 2013 doi: 10.1109/ICCNC.2013.6504102

[10] Portico RTI - www.porticoproject.org

[11] Sztipanovits, J., and Karsai, G. 1997. "Model-Integrated Computing", IEEE Computer, 30(110-112)

[12] de Laura, J., and Vangheluwe, H., 2002. "AToM3: A Tool for Multi-formalism and Meta-Modeling", Lecture Notes in Computer Science, 2306 (174-188).

[13] Tolvanen, J.P., and Lyytinen, K. 1993. "Flexible Method Adaptation in CASE. The Metamodeling Approach", Scandinavian Journal of Information Science, v5 n1 (71-77).

[14] Cook, S., Jones, G., Kent, S., and Wills, A. 2007. "Domain-specific Development with Visual Studio DSL Tools", Addison-Wesley Professional

[15] The Eclipse Foundation – www.eclipse.org

[16] Modelon, Inc. – www.modelon.com

[17] JFMI: A Java wrapper for the Functional Mockup Interface – www.ptolemy.eecs.berkeley.edu/java/jfmi

[18] OMNeT++ - www.omnetpp.org

[19] DARPA Adaptive Vehicle Make Program – www.darpa.mil/Our_Work/TTO/Programs/Adaptive_Vehicle_Make__(AVM).aspx