Simulation Integration Platforms for Cyber-Physical Systems

Himanshu Neema
Janos Sztipanovits
himanshu.neema@vanderbilt.edu
janos.sztipanovits@vanderbilt.edu
Vanderbilt University
Nashville, TN, USA

Cornelius Steinbrink
Thomas Raub
cornelius.steinbrink@offis.de
thomas.raub@offis.de
OFFIS – Institute for Information Technology
Oldenburg, Germany

Bastian Cornelsen
Sebastian Lehnhoff
bastian.cornelsen@uni-oldenburg.de
sebastian.lehnhoff@uni-oldenburg.de
University of Oldenburg
Oldenburg, Germany

ABSTRACT

Simulation-based analysis is essential in the model-based design process of Cyber-Physical Systems (CPS). Since heterogeneity is inherent to CPS, virtual prototyping of CPS designs and the simulation of their behavior in various environments typically involve a number of physical and computation/communication domains interacting with each other. Affordability of the model-based design process makes the use of existing domain-specific modeling and simulation tools all but mandatory. However, this pressure establishes the requirement for integrating the domain-specific models and simulators into a semantically consistent and efficient system-of-system simulation. The focus of the paper is the interoperability of popular integration platforms supporting heterogeneous multi-model simulations. We examine the relationship among three existing platforms: the High-Level Architecture (HLA)-based CPS Wind Tunnel (CPSWT), MOSAIK, and the Functional Mockup Unit (FMU). We discuss approaches to establish interoperability and present results of ongoing work in the context of an example.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Modeling and simulation**; • **General and reference** → *Cross-computing tools and techniques.*

KEYWORDS

Cyber-physical systems, simulation integration platforms, model-integration, tool-integration, execution-integration, modeling and simulation, distributed simulation, co-simulation

ACM Reference Format:
Himanshu Neema, Janos Sztipanovits, Cornelius Steinbrink, Thomas Raub, Bastian Cornelsen, and Sebastian Lehnhoff. 2019. Simulation Integration Platforms for Cyber-Physical Systems. In *Design Automation for CPS and IoT (DESTION '19), April 15, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3313151.3313169

DESTION '19, April 15, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6699-1/19/04…$15.00
https://doi.org/10.1145/3313151.3313169

1 INTRODUCTION

Cyber-physical systems (CPS) are engineered systems where functionality emerges from the networked interaction of physical and computational processes [17]. Design of these systems requires the integrated use of domain-specific abstractions and analysis methods that have been developed over the past decades in disparate areas of computing and networking as well as physical systems engineering such as mechanical, thermal, electrical, electronic, hydraulic, and other domains.

Highly diverse CPS components, multi-physics behavior domains and complex interactions make it impossible for a single simulation tool to satisfy all of the modeling and analysis requirements [29] [15]. Therefore a suite of simulation tools, each simulating some aspects of a particular CPS, need to be used for system level CPS simulation. For example, a communication network can be simulated by a variety of well-known open-source simulation tools such as OMNeT++ [31] and NS-3 [11] or by commercial simulation tools such as OPNET [12]. Physical processes in plants can be simulated by Dymola [8] or MATLAB-Simulink [3]. These simulation tools also provide a large set of curated model libraries that enables efficient construction of good models using them. Thus, by necessity, we have to integrate in a semantically precise manner different modeling languages and simulators to perform a system-of-systems level holistic analysis of CPS.

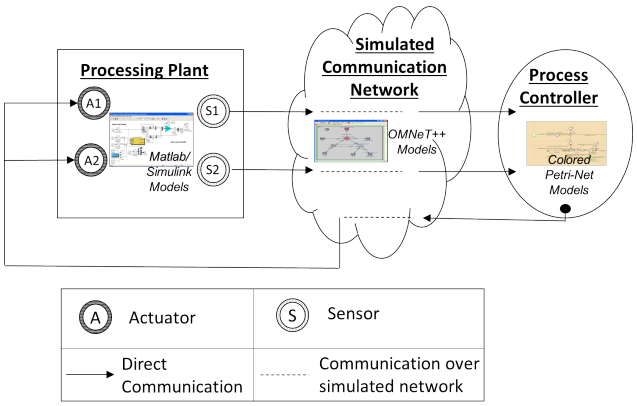

Figure 1: A Network-Controlled Processing Plant

Fig. 1 shows a simple illustration of three aspects of a processing plant. The plant is simulated using MATLAB-Simulink [3] software, the event-based controller is simulated using Colored Petri Net

(CPN) [18], and the communication network connecting the plant and controller is simulated using OMNeT++ [31]. Integrated evaluation of the three simulators can show the effects of network delays on plant operation. It can also answer questions about the resilience of the processing plant to physical faults or cyber-attacks.

Integrating simulations tools in a semantically precise manner is highly challenging. These diverse simulation tools differ heavily in the types of models and modeling languages they use, their models of computation, their implementation language, interfaces, and behavioral semantics. In addition, industry heavily uses proprietary simulators that make *ad-hoc* approaches for their connection ineffective. The simple input-output connection among simulators running on a distributed computing platform completely overlooks the fundamental integration requirements such as precise time coordination and distributed object management for data exchange.

This paper describes a conceptual integration framework that we have developed over last several years [26] [14] [15]. The framework incorporates three distinct integration platforms: the model-integration platform (MIP), the tool-integration platform (TIP), and the execution-integration platform (EIP). We focus mainly on model and tool integration platforms, which represent the core part of model-based integration approaches. In Section 2 we summarize the specific services integration platforms need to provide and discuss the overall framework and the relationship among the MIP, TIP and EIP platforms. In Sections 3, 4, and 5, and we respectively review three specific integration platforms, the Cyber-Physical Systems Wind Tunnel (CPSWT) [14] [15], the MOSAIK platform developed for smart-grid co-simulation [10], and the Functional Mock-up Unit (FMU) [6]. In Section 6 we discuss the need for interoperability among various simulation integration platforms. We present our ongoing work to enable interoperation among the above integration platforms in Section 7. Finally, Section 8 concludes the paper and highlights directions for future work.

2 SIMULATION INTEGRATION CHALLENGES

Real-world CPS are frequently large-scale system-of-systems that are built using many different subsystems. These individual subsystems often exhibit dynamic behavior and utilize discrete- or continuous-time models for their simulations. During simulation, these dynamical models compute system behavior over time, so interactions among simulated subsystems need to support both data exchange and time-based coordination.

2.1 Common Requirements in Distributed Simulation and Experimentation of CPS

The common challenge for simulation integration platforms is that the suite of domain specific simulation tools that need to be supported is not fixed. Depending on the type of CPS to be analyzed, specific goal of the analysis, and the experimental scenarios targeted, the tool configuration varies. Listed below are typical requirements that simulation integration platforms need to satisfy.

- ***Time Management***: In order to maintain logical and temporal consistency among time-dependent simulations, it is required to manage timed execution of simulation models, synchronize time across simulators, coordinate their time advancement through a configurable time advancement protocol, allow time-stamped events that are shared and processed in a time-dependent manner according to a pre-determined event delivery and handling protocol, permit across simulators different time-scales and time-resolutions (potentially even dynamically varying), and support for simulation execution in a real-time or as-fast-as-possible manner.
- ***Distributed Object Management***: Rich interaction models are necessary for distributed, multi-model simulations. Data exchange may occur via one-off interactions or through stateful shared data-structures. Both mechanisms should support time-stamped and untimed data updates, reliable and best-effort data delivery and processing rules and methods. Support also is needed for ensuring a desired Quality of Service (QoS) of the physical network used for data exchanges [13]. In addition, often some simulators join and leave the executing integrated simulations multiple times during run-time, thus requiring dynamic discovery and tracking of the simulators, maintaining current data model types for exchanged data, and updating the assignment of data producers and consumers for the updated data model among the updated set of currently executing and integrated simulators.
- ***Coordinated Simulation Orchestration***: For temporal consistency, all the integrated simulators must start executing simultaneously. User control of simulation execution and experimentation with different *logical* and *physical* start times is also needed. In addition, these complex simulation experiments require configuring many parameters and settings. Such tasks are usually accomplished using a dedicated *simulation manager* component [15].
- ***Integration with Hardware, Humans, and Existing Systems***: Many physical phenomena are better suited to execution in the physical hardware than simulation for reasons such as unavailability of high-fidelity models or significantly lower computational performance (e.g. software-defined radios) [23]. In addition, large systems and services (e.g. experiment infrastructure or remote laboratories) may need to be integrated directly with simulations. Further, these simulations often need integrated modeling of human interactions (e.g., machine operators), human organization models, operational workflows, and decision-making processes. As the physical hardware and humans operate in real-time, the integrated simulations must also execute in real-time. Many other challenges arise due to synchronization, network delays, authentication and authorization, and other operational factors such as availability and costs.
- ***Communication Network Simulation and Emulation***: In CPS, networking is a key component that cuts across many simulations. Thus, integrated CPS simulations invariably require simulation (or emulation [33] using physical devices) of the communication network. This allows for studying CPS security and resilience against effects such as network delays, packet drops, and cyber-attacks.
- ***Scenario-Based Experimentation***: This is needed for evaluating the systems with a range of run-time variability in the component models and in operation scenarios [23] [25].

- ***Compositionality and Semantic Interoperability***: For simulation interoperability, binding them using same inputs and outputs is highly inadequate. The simulation models and tools make internal assumptions behind their interfaces that must be respected for their semantically accurate composition. For example, simulation exchanging voltage value might use different measurement units and assume different update frequencies. Thus, each simulation has a contract for its input-output ports that specifies the conditions that must be satisfied for meaningfully interacting with it. The contents of this contract must be defined by each co-simulation platform, and a mechanism must be implemented to ensure that two simulations can interoperate at this contractual level when attempting to perform a combined experiment.
- ***Rapid Synthesis of Integrated Simulations and Experiments***: Important CPS simulation applications such as automated design-space exploration [22] and automated vulnerability analysis [19] require changing modeling abstractions during the search process that implies changing the suite of included simulators. Manual integration of needed simulations is inefficient, error-prone, and inflexible in these cases. However, automated integration of simulations [14] using integration models is a hard problem requiring significant configurable tools and computation infrastructure [5].
- ***Broader Usability***: These tools are the front-end that a CPS researcher uses for experiments and so these must be usable by a broad range of users, some of whom have limited background in co-simulation, distributed systems, and programming. Developing complex CPS co-simulation requires significant expertise, thereby requiring a suite of support tools to improve the user experience and make the benefits of co-simulation available to other researchers. These include: tools for automated and managed compilation of updated simulation sources and models and packaging compiled artifacts for their run-time use; tools for deploying simulations and provisioning resources for them across different computational platforms such as servers, virtual machines, or a computation cloud; tools for monitoring and controlling simulation runs and reporting run-time events; and tools for comprehensive analysis of experiment results.
- ***Reusable Component Libraries***: Curating and validating simulation models and experiments are time-consuming and expensive. Reusing parts of simulations in new experiments and different use-cases is an important goal. However, CPS simulation tools use many different models with different meta-data that would need to be captured in a consistent manner. The challenge is in coming up with a generic component model that is versioned and configurable and enables packaging componentized models and experiments for reuse in different experiment context with reasonable confidence.

2.2 Framework for Horizontal Integration of Simulators

Existing simulation tools are usually tied to specific domains such as power flow analysis or transportation networks. These simulation tools are vertically integrated and incorporate modeling languages, modeling tools, model libraries, simulation engines and interfaces for getting access to results. However, system-of-systems simulations require horizontal integration (see Fig. 2) that cut across these vertically isolated simulation tools and models and facilitate cross-domain interactions. Our framework incorporates three horizontal integration platforms: (1) Model Integration Platform, (2) Tool Integration Platform, and (3) Execution Integration Platform.

Simulation tools integration for co-simulation have several well-established architectures. The High Level Architecture (HLA) [1] (IEEE Standards Association 2016) is a standardized architecture for distributed computer simulation systems. The Functional Mock-up Interface (Modelica Association 2014a) for co-simulation is a relatively new standard [2] targeting the integration of different simulators. In spite of the maturity and acceptance of these standards, there are many open research issues related to scaling, composition, large range of required time resolution, hardware-in-the-loop simulators and increasing automation in simulation integration. Execution Integration Platforms for distributed co-simulations are shifting toward cloud-based deployment, developing simulation-as-a-service use model via web interfaces, and increasing automation in dynamic provisioning of resources as required.

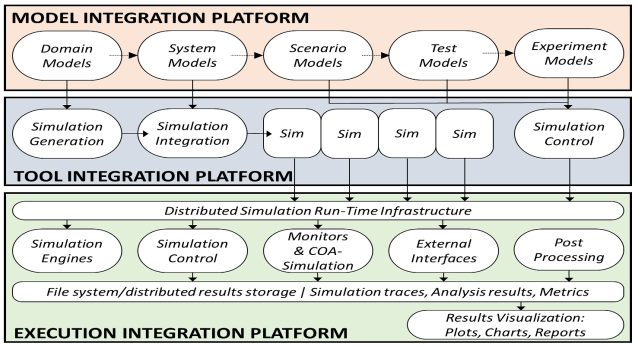

Figure 2: Horizontal Integration Platforms for CPS Simulations

The key components of the three integration layers are:

(1) ***Model Integration Platform (MIP)***: The function of the model integration platform is to model interactions among a wide range of heterogeneous domain models in a semantically sound manner. The integration model captures the high-level system-of-systems architecture where subsystems may use different domain-specific modeling languages (DSML) [30] and simulators. One of the major challenges is semantic heterogeneity of the constituent systems and the specification of integration models. There are several approach for supporting the integration of heterogeneous models.

Modeling Language Embedding: Embedding requires an injective, structure preserving mapping between one or more DSML and a host language. Accordingly, the semantic domain of the host language must be rich enough to provide a common semantic domain for all DSMLs. For example, the continuous time (CT) semantics of Simulink [3] can embed the discrete event (DE) semantics of CPN [18] and

OMNeT++ [31], but mapping many of their domain-specific languages constructs onto Simulink constructs would be impractical. Some variant of model embedding is facilitated by the 'external' blocks allowed in some modeling languages (e.g. Modelica's 'external' function calls [6], FMU import [8], and Simulink's S-function interface [3]) but they require suppressing many details of an external model, transform it into a simple construct in the host language, therefore they cannot be considered model embedding in the usual sense.

Formal Modeling Language Composition: There have been established methods for the formal composition of DSMLs in an algebraic/logic framework. The approach introduces a range of composition operators (includes, restricts, extends, as, pseudo-product '*', pseudo-coproduct '+') with appropriate semantics. These composition operators have been introduced in the FORMULA tool [16] developed by Ethan Jackson at Microsoft Research. While precise and tool supported, practicality of a full formal treatment is restricted by the absence of a formal, FORMULA-based specification of the semantics of the constituent modeling languages.

Model Integration Language: In most multi-modeling problems, the goal of model integration is to capture the interaction among physical or computational objects modeled using different DSMLs. The interaction might have behavioral, structural or conceptual meaning. If the semantics of the interaction is restricted only to some shared aspects of the semantics of the individual modeling languages, the problem can be solved effectively by specifying a model integration language that includes the specification of a semantic interface for the individual modeling languages and the specification of integration constructs that are not part of either of the integrated modeling languages but support the integration of the models across the semantic interfaces. For example, in Fig. 1, the purpose of the model integration is the coordination of timed behavior of physical processes, discrete controllers and communication networks. The required interaction semantics is discrete event and the data semantics is based on a distributed (but partial) data model that need to be established for the scenario. Consequently, model integration requires the specification of a relatively narrow semantic interface for the individual DSMLs, a Model Integration Language which is built on these semantic interfaces and extended with integration specific constructs such as timed interactions and data models.

(2) ***Tool Integration Platform (TIP)***: The function of the tool integration platform is to ensure that the execution of simulations can be synchronized by a distributed global time clock (or with the real-time clock). Furthermore, the events generated by the individual simulators can be used for coordinating event-driven interaction among objects controlled by different simulators. TIP needs to also ensure that data can be routed among the simulators under the time constraints required by the progress of the distributed global clock. Integration platform for simulation tools is a well developed area with several known standards.

(3) ***Execution Integration Platform (EIP)***: The simulation tools used for large system-of-systems are usually special-purpose and complex and require tool-specific computational resources; and tools and methods to control, monitor and supervise their execution. Automated experimentation using integrated simulation and analysis through high-level evaluation scenarios require tools and services that orchestrates the distributed simulation in a coordinated manner on the available computational resources such as servers, virtual machines, and cloud. The MIP includes tools and methods that enable dynamic experiment configuration and deployment on computational infrastructure as well as execution and monitoring of integrated simulation experiments and collection and processing of generated experiment results.

The above three horizontal integration platforms provide a conceptual framework for approaching simulation integration challenges that occur in all system-of-system simulations. The tools and services provided by each of these integration platforms should be domain-independent and vendor neutral. The instantiation of these tools and services in a domain-specific way is accomplished by model parameterization and configurations. Together, these three integration platforms enable robust and extensible frameworks for rapid synthesis of heterogeneous simulation integration and automated experimentation. In the sections below, we detail the key elements of model and tool integration platforms in three specific architectures, the CPSWT [14] [15], MOSAIK [10] and FMI [6].

3 CYBER-PHYSICAL SYSTEMS WIND TUNNEL (CPSWT)

Over the past decade, Vanderbilt has developed a model-based, multi-model simulation integration platform called the Command and Control Wind Tunnel (C2WT) [15]. The initial motivations came from the need of experimental evaluation of command and control architectures in different mission contexts [14]. Later, the system was extended to the simulation-based study of networked controllers [33], cyber-attacks on CPS [7], and became the foundation for virtual prototyping in the OpenMETA design automation tool chain [29] under the name of CPSWT. Driven by the needs of studying resilience of large-scale infrastructures such as traffic control [19], railway control [7], and transactive energy [27], in 2014, C2WT was adopted by NIST and, working jointly with Vanderbilt, C2WT evolved into NIST's CPS simulation platform (UCEF) [9]. In this section, we provide an overview of CPSWT model and tool integration platforms using the conceptual framework described before. Fig. 3 shows a simplified view of the two platforms and their connection using the processing plant example in Fig. 1.

3.1 Tool Integration Platform in CPSWT

We adopted the High Level Architecture [1] as the basis for the Tool Integration Platform. The HLA provides a set of standard services for configuring, executing, and coordinating the system-of-system distributed simulations. CPSWT uses an open-source implementation of HLA called the Portico [4] Run-Time Infrastructure (RTI), which is written in Java language and also supports full bindings for C++ language, and is fully compliant with HLA 1.3 and implements the latest version HLA-1516e.

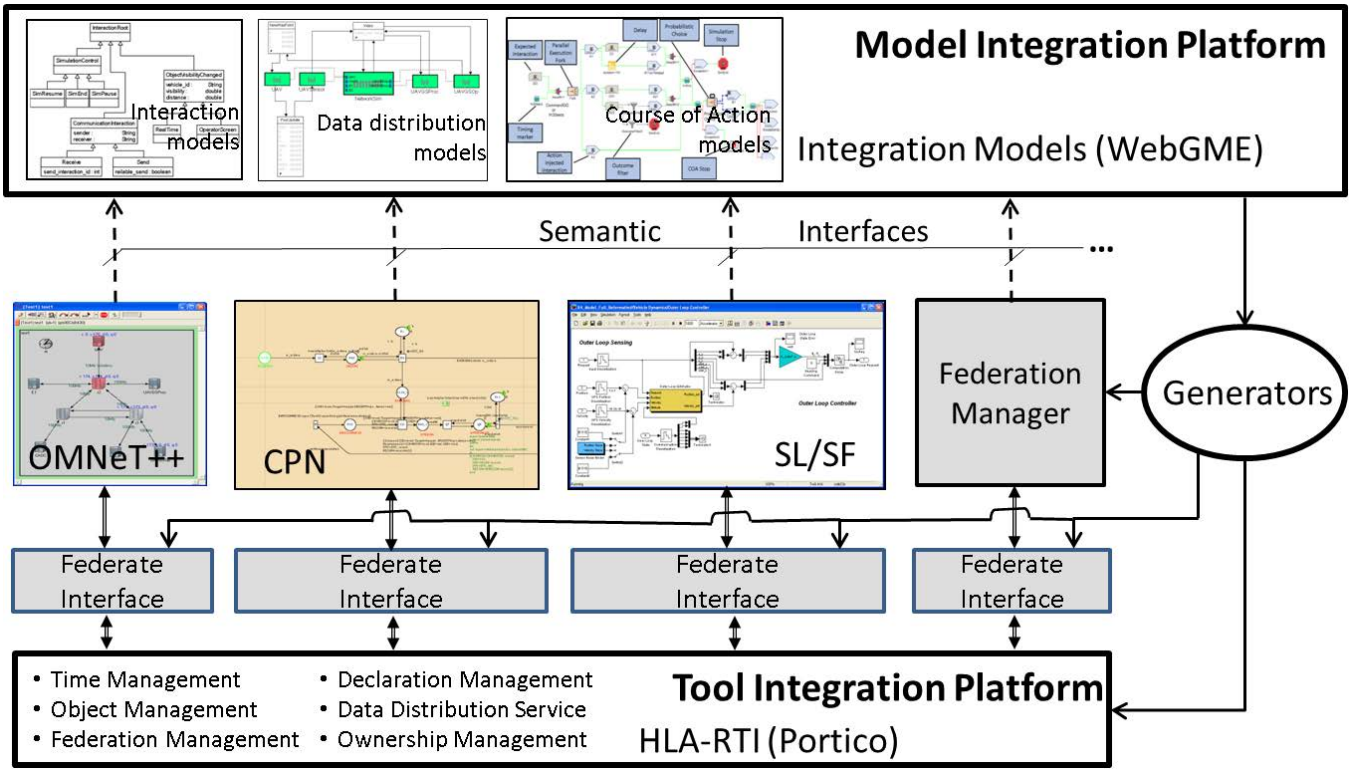

Figure 3: Model and Tool Integration Platforms

We have selected HLA because it is a well known mature standard, and offers flexible services for advanced and configurable time management, data distribution, object management and object migration and overall simulation management performed by a Federation Manager. Coordination and data distribution across the individual simulators is performed by the HLA-RTI via the Federate Interfaces. The semantic interface between the modeling languages of the individual simulators is provided by the HLA standard. The integration models between the models of each simulators and the RTI are used for generating the corresponding HLA Federates that directly supports the construction of complex multi-model simulations. It is not our purpose to describe the details of the standard HLA services, because they are extensively documented in the standard and in many tutorials. A frequently mentioned argument against HLA is that it is complex, heavy weight and extremely hard to use. However, the model-based integration framework we developed in CPSWT largely alleviates these concerns:

- A Model Integration Platform (Section 3.2) enables the auto-generation of Federate Interfaces and the Federation Manager from a high-level system-of-system model specified in a graphical modeling environment, WebGME [21].
- The integration architecture allows the separation of the fine-grain internal interactions in the domain specific simulators and the federation-level interactions managed by HLA-RTI.
- Decomposition of large dynamic models into distributed, interacting simulation models based on differences in system dynamics offers an effective way to manage highly different time resolutions [24].

3.2 Model Integration Platform in CPSWT

The Model Integration Platform takes full advantage of the configurability of the HLA services. The Model Integration Language incorporates three sub-languages for *interaction modeling*, *data distribution modeling* and *course of action modeling*. The level of abstraction in these models can be adjusted via the definition of the semantic interfaces for the individual simulators. The integration models are built by means of our metaprogrammable graphical editor, WebGME [21] that facilitates the customization of the modeling environment by means of metamodeling [30]. All of the sub-system models are defined by using the native modeling languages and tools of the integrated simulators, therefore the integration models are restricted to the modeling of system-of-system level interactions.

The *course of action models* represent a simulation scenario with modeling elements such as events, event conditions, durations between events, alternative scenario paths, and synchronization points [23] [25]. With scenario models complex experiment scenarios such as what-if situations or alternative evaluation paths can be modeled. As Fig. 3 shows, the integration models are used by a suite of generators (implemented as plug-ins for the WebGME model builder) to synthesize the federate interfaces and the federation manager.

4 MOSAIK SMART-GRID CO-SIMULATION PLATFORM

The MOSAIK framework has been designed especially for the co-simulation of CPS in the energy domain (CPES, e. g. Smart Grids) [10] by OFFIS and University of Oldenburg. True multi-domain system analysis is a rather new topic in the energy domain, so the established tools are so far not designed for integration with other systems. Consequently, as shown notionally in Fig. 4, the MOSAIK design is focused on the coupling of strongly heterogeneous simulation tools from various different disciplines while imposing as little structural requirements on the tools as possible.

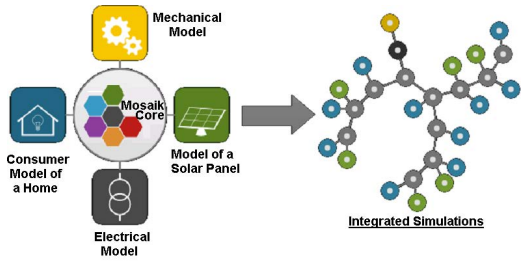

Figure 4: MOSAIK Smart-Grid Co-Simulation Framework

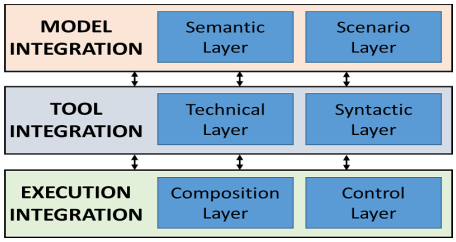

Figure 5: MOSAIK Conceptual Architecture

Fig. 5 shows the original conceptual architecture of MOSAIK. The framework directly supports key aspects of the model-integration, tool-integration, and execution-integration as described previously. Different requirements of integration are addressed in MOSAIK through a set of distinct functional layers.

Model Integration in MOSAIK is accomplished using the *Semantic* and *Scenario* layers. The semantic layer describes a reference data model that is agreed for exchanging data among the integrated simulators. In addition, the semantic layer also specifies interaction ports that are used to describe data flows among the simulators and support tools for verifying composability. The scenario layer captures the actual configuration of integrated simulation models. In addition to integration models, the scenario layer also supports specification of co-simulation scenarios and a rule-based method to validate them based on semantics described in the semantic layer.

Tool Integration in MOSAIK is accomplished using the *Technical* and *Syntactic* layers. The technical layer provides the core physical infrastructure to execute integrated simulations. This includes the computational and networking hardware and supporting tools to orchestrate the distributed simulations. The syntactic layer specifies the technical interfaces needed to be implemented for integrating simulation tools.

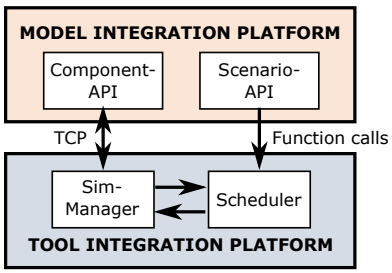

Figure 6: Components Implemented in MOSAIK

Execution Integration in MOSAIK is accomplished using the *Composition* and *Control* layers. The composition layer provides tools for validating the integrated simulations and executing them based on the specification in the scenario layer. The control layer provides the *Control-API*, which specifies the interfaces for accessing and updating state variables of the simulated models.

The encapsulation of distinct functions of model-integration, tool-integration, and execution-integration into separate functional layers allows MOSAIK to support experimentation with different algorithms and implementations in each of the layers.

4.1 Current MOSAIK Implementation

The implementation of the MOSAIK concept has undergone some changes since its original drafting. More precisely, some of the coupling validation aspects associated with execution-integration have been ported into an individually developed toolbox. The resulting reduced scope of the MOSAIK core leads to higher modularity and easier deployment and handling by new users. Fig. 6 describes key components of the current MOSAIK implementation. As shown, the system consists of three major parts: The algorithmic system core, the *Component-API* for integration of simulation tools, and the *Scenario-API* for the creation of co-simulation experiments. All these framework components are purely based on the Python programming language to ensure a seamless software installation without a multitude of dependencies.

(1) ***Component-API***: The communication between MOSAIK and the simulation tools via the *Component-API* is based on the exchange of JSON encoded messages through TCP sockets so that any tool can be integrated that supports these two concepts. Coupling of models with different temporal resolution can efficiently be solved via data buffering in the MOSAIK core. The communication between MOSAIK and a simulation tool is defined through a minimal set of four message types called `init`, `create`, `step`, and `get_data`. The first two messages are used during experiment description and allow MOSAIK to send parameter values and other configuration data to a given tool. In its response, the simulation

tool provides MOSAIK with static information about the types and numbers of models provided by the tool. The `step` and `get_data` messages are called by the MOSAIK scheduling algorithm during simulation execution to send input data to a model, advance it in time, and request output data.

(2) ***Scenario-API***: The *Scenario-API* allows users to set the parameters of and the data exchange between simulation models to establish executable CPES co-simulation setups. The data handles for access to the models is provided via the `init` and `create` messages. The data that is exchanged with a model entity is divided into two types: *parameters* and *attributes*. Parameters are used to configure model behavior at the beginning of the simulation process, e. g. parameters may be PV panel size or efficiency. Attributes, on the other hand, are used to establish data exchange connections between models from different simulation tools, i. e. an attribute of a model may serve as a data input port, a data output port, or both. Value assignment to parameters as well as attribute-based model connections are realized via the interaction of the Component-API and the Scenario-API. It should be noted, however, that little domain-specific constraints are included to avoid inflexibility towards heterogeneous tools. Therefore, it is a task of the user to account for data consistency between coupled models.

(3) ***Scheduler***: The scheduler module is responsible for the temporal and logical correctness of data exchange between the simulation models. It maintains data flow dependencies between integrated tools, as defined by the user, and keeps track of the simulation time of each tool. As the central co-simulation executor, the scheduler can trigger or postpone the next simulation step of a given model, depending on the availability of its required input data.

(4) ***Simulation Manager***: The simulation manager establishes TCP connections among the integrated tools and handles the JSON message exchange with them. It supports the scheduler module by providing data to and requesting data from integrated tools via the `step` and `get_data` messages.

In summary, the MOSAIK framework is a CPS co-simulation platform that directly addresses the basic challenges of simulation integration. It uses its lightweight setup to boost flexibility and its centralized structure to ensure usability. For the integration of simulation tools, a number of technologies are supported. While the basic integration is based on TCP sockets and JSON encoding, several language-specific adapters have been developed to reduce implementation overhead for interfaces.

5 SIMULATION INTEGRATION USING FMI

We consider the Functional Mock-up Interface (FMI) [6] [2] as it is a widely used co-simulation standard. The key feature of the FMI standard is that it provides a standardized template for encapsulating a dynamic system's simulation model. While assuming a generic solver that executes a dynamical system model, the standard provides standard interfaces for interacting with the model. The generic solver executes the model and computes its behavior.

The FMI standard defines a set of functions that support configuration, setup, and initialization of models, as well as their dynamic use for access to and modification of model variables. The models can keep the implemented behavior in binary format (thus preserving IP) while providing an access and control mechanism that supports these standard functions. The implemented executable model based on FMI is called a Functional Mock-up Unit (FMU). An FMU is a zip-file containing a meta-data description XML file and a shared binary that implements its behavior and FMI interfaces.

5.1 FMI Model-Exchange and Co-Simulation

The FMI standard contains two key parts, viz. FMI for Model Exchange (FMI-ME) and FMI for Co-Simulation (FMI-CS). FMI-ME specifies a standard mechanism for the distribution of a dynamic system model. The model is described by differential, algebraic and discrete equations with events based on model's time, state, and step. The distributed model could be in the form of generated C-Code that can be directly linked to other models in a simulation environment. FMI-CS specifies a standard mechanism for co-simulation of two or more simulation tools. The co-simulated models exchange data at discrete time-points, and execute independently at other times. The FMUs may or may not embed a solver for their execution. Fig. 7 shows *FMI-CS for models*, which allows coupling FMUs that do not embed a solver using a *'master algorithm'* that coordinates their co-simulation for data-exchange and time-synchronization. However, as the FMI standard does not specify the master algorithm, its design and implementation is simply left to individual simulation tool providers. The variation *FMI-CS for simulation tools* allows coupling FMUs that do embed a solver. These FMUs are interfaced using an FMI Wrapper that implements the standard FMI functions. Here, the FMUs are solved by an external numerical integration solver (supplied by the simulation environment).

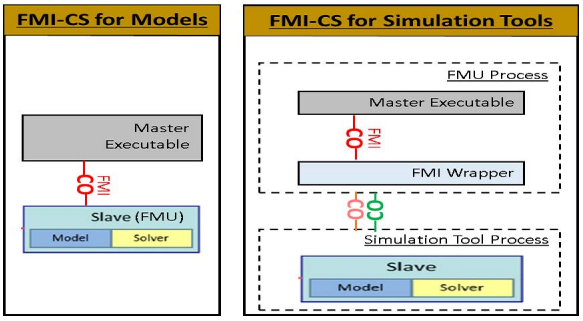

Figure 7: FMI for Co-Simulation

5.2 Challenges with FMI Co-Simulation

Many tools exist, such as Dymola [8] and MATLAB-Simulink [3], that support creating (and importing) FMUs and provide tool-specific implementation of the master algorithm. However, as the FMI standard simply focuses on standardizing the dynamical system model's interface, many of the fundamental simulation integration challenges (see Section 2.1) are not fully addressed. In particular, time management and distributed object management, that are crucial in large-scale distributed simulations, are not specified. Thus, different simulation-tool-specific implementation of the master algorithm can lead to different behavior in the co-simulation.

6 INTEROPERABILITY OF SIMULATION INTEGRATION PLATFORMS

As described in Section 2.1, integrating CPS simulations is a challenging task. Different simulation integration platforms use different trade-offs of flexibility, usability, and performance in addressing these challenges. Thus, no platform can achieve high quality in all such aspects. For example, tool performance can be optimized regarding the constraints of a given application case, but this decreases the tool's flexibility. On the other hand, if performance optimization is done for multiple applications to boost flexibility, this increases the tool complexity, likely decreasing its usability. Simply put, every simulation integration platform has a given niche. Given trade-offs between different platforms and the overall complexity of the CPS domain, two major goals can be identified for enabling interoperability among simulation integration platforms:

- **Synergy:** Interoperation with other platforms can extend the capabilities of individual platforms. In this case, each platform governs a complex setup that benefits from its specific qualities in terms of flexibility or performance. Platform coupling then allows for even more complex simulation experiments without limitation due to platform specifications.
- **Reusability:** Interoperability can improve reusability and thus decrease implementation effort. One way to combine two complex simulation setups, based on different integration platforms, is to directly extend the setup of one with the tools of the other. However, that requires a significant effort on interfacing. Platform interoperability can provide an easier way for coupling existing simulation setups.

The above two goals also imply other potential benefits of platform interoperation. For example, a synergy between platforms might also lie in coupling of different time scales or levels of granularity. However, expanding further on this topic is likely beyond the scope of this paper. Therefore, we focus on two practical use cases that exemplify the interoperability of simulation integration platforms.

Use case 1 is based on concepts like energy communities or *microgrids* [20]. These potentially autonomous systems are typically defined by local power grid infrastructures and a high number of distributed energy resources (DER) that are controlled to improve the power consumption in the system, e. g. price efficiency for the local consumers. However, this depends strongly on the local energy mix, i. e. the number and types of DER, the types of consumers, and the availability of storage capabilities. Furthermore, interaction between the system and its surrounding region may be influenced by transmission grid feed-in, weather phenomena, the communication infrastructure and more. The proposed use case involves a simulation of the microgrid using MOSAIK and a simulation of the information and communication infrastructure (ICT) as well as environmental and large-scale influences using CPSWT. This setup is an example for a synergy in the interoperation between the two platforms. On the one hand, the orchestration of the microgrid setup benefits from the centralized design of MOSAIK. Also, the logical data exchange requirements between microgrid components is unlikely to change, regardless of the numbers and types of integrated DER. Therefore, MOSAIK allows rapid testing of a high number of microgrid layouts and DER compositions with only minimal setup changes via the Scenario-API. On the other hand, CPSWT provides a greater flexibility of tool coupling due to its HLA-based structure. This allows for easy integration of special events, dynamic changes in the coupling, and so on. This way, extreme weather events, communication delays as well as faults or attacks in the ICT can be easily introduced into the testing.

Use case 2 follows a similar design as the first one, but implies a different implementation (see Section 7). In this case, the majority of components of a Smart Grid system are coupled via MOSAIK. This involves the complete power grid simulation, all DER, the control infrastructure, and energy markets. However, the communication infrastructure is realized via a simulation tool integrated in CPSWT. Through a composition of MOSAIK and CPSWT, communication dynamics including delays or packet-loss can be integrated into the overall Smart Grid simulation. On the one hand, this use case reflects the synergy aspect of platform interoperation since the CPSWT flexibility is more suitable than MOSAIK for realizing discrete-event dynamics that lie at the core of communication simulation. On the other hand, the use case shows the reusability aspect of platform interoperation since CPSWT is already strongly integrated with communication simulation while MOSAIK displays established integration of DER and controller simulation.

7 INTEROPERABILITY APPROACHES

Interoperation among simulation integration platforms can be designed in a modular way by employing a hierarchical approach. This means that one platform (including all its integrated tools) is integrated into the other platform in the same or a similar fashion as standard simulation tools. In the following we describe our approach for bridging these platforms: (i) FMU co-Simulation with CPSWT and MOSAIK, (ii) MOSAIK co-simulation with CPSWT.

7.1 FMU Co-Simulation with CPSWT or MOSAIK

The CPSWT platform supports a variety of heterogeneous simulation tools to be integrated as part of the system-of-systems simulations. In addition, CPSWT also supports interfacing with components based on the Functional Mock Interface (FMI) standard [6]. FMI is a well-known co-simulation standard for model reuse and IP protection. The FMI standard provides a template for encapsulating dynamic system's simulation models. However, FMI does not specify how to execute FMI components in a coordinated manner. In addition, FMI also does not support sophisticated time management services and distributed object management that are supported by HLA. Therefore, we extended the CPSWT platform to integrated FMI components (a.k.a. Functional Mock-up Units (FMUs)) as HLA federates. As shown in Fig. 8, for every FMU to be integrated with other HLA-federates, we automatically generate a FMU HLA Wrapper that provides both the interconnection with HLA and access and control of the FMU. We covered the FMU integration in CPSWT in detail in [24].

We also showed in [24] a method to partition a dynamical model into different sampling rate groups using FMUs. This allows executing parts of the model with slower rate dynamics at larger step-sizes, thus leading to better overall simulation performance.

Similar to the coupling between FMI and CPSWT, FMUs can be integrated into MOSAIK co-simulation processes. This is possible via a mapping between the model description and methods specified by

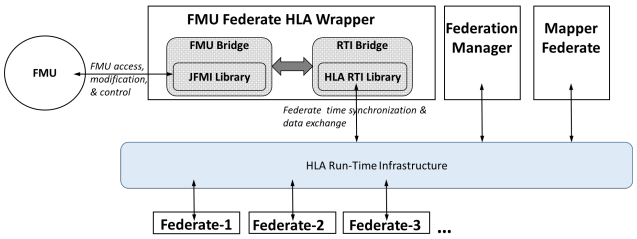

Figure 8: Integration of FMI Components in CPSWT

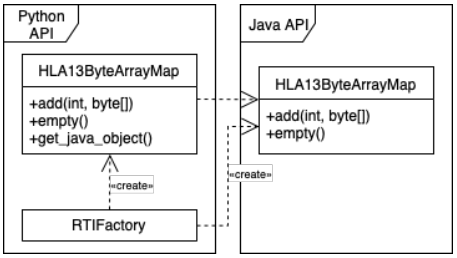

Figure 10: Delegation from Python-API to Java-API

the FMI standard and those described by the MOSAIK Component-API [28]. In case of a coupling, an FMU is controlled actively by MOSAIK acting as a master algorithm. The coupling can be directly achieved for FMUs following the FMI for Co-Simulation standard. For those units following the FMI for Model Exchange standard, additional solver capabilities have to be provided by MOSAIK. Tools like the *FMI++ library* [32] help to integrate such capabilities in an interface module.

7.2 MOSAIK Co-Simulation with CPSWT

Since the MOSAIK core is based on the Python programming language, a Python-API has been developed for CPSWT federates. The API is based on the *Py4J* library[1] that creates a bridge between Python and Java and enables calls to Java code from Python. This way, the API can wrap the Java code of the CPSWT *federate ambassador* interface and a call from the Python-API will lead to corresponding call of the Java-API. The delegation and inheritance pattern is used in this wrapping. In other words, a Java interface may be implemented in the Python-API through inheritance (Fig. 9). This is enabled by Py4J. The implemented Python class can be passed in a method call of the Java-API. Since the Portico HLA-RTI works asynchronously, typically callbacks with Python code are passed to the Java-API. Similarly, the Python-API can delegate requests to the Java code (Fig. 10). If non-primitive data types are to be received by the Python code, Java objects are returned that are accessible through access modifiers provided by Py4J.

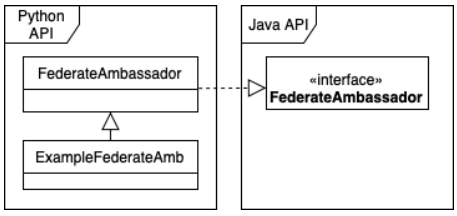

Figure 9: Python-API for Portico using the Java-API

These APIs enable integration of an executable MOSAIK setup as a CPSWT federate. This federate must contain the MOSAIK core, integrated simulation tools, and an executable scenario script. The scenario script defines how the integrated simulation tools and models are coupled with each other, and allows execution of the MOSAIK scheduling algorithm. Finally, data exchange between the MOSAIK co-simulation and the CPSWT federates has to be enabled. The easiest way to realize this is by implementing a dedicated data exchange component via the Component-API. This module interacts with the MOSAIK core like any other integrated simulation tool, but all data obtained from MOSAIK can be forwarded to CPSWT via the federate ambassador. Similarly, the data may be obtained from other federates and forwarded to MOSAIK. The overall composition concept is outlined in Fig. 11. This represents a reasonable implementation of use-case 1 described in Section 6.

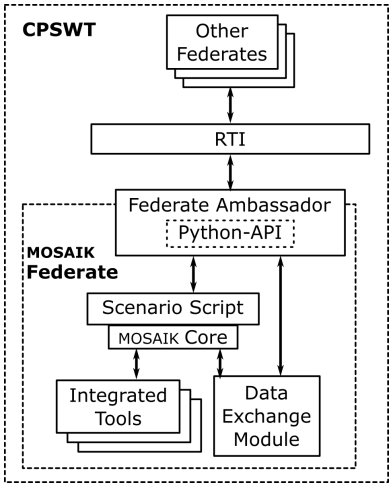

Figure 11: Composition concept with MOSAIK federate.

8 CONCLUSIONS & FUTURE WORK

Cyber-physical systems (CPS) are inherently highly complex to model and analyze as their behavior emanates from tight coupling of physical and computational components. This necessitates the use of domain-specific modeling languages and tools for their analysis. Simulation-based evaluation is a well-accepted method due to apparent complexity in formal analysis of real-world complex CPS.

CPS component models are highly diverse, represent many different multi-physics domains, and are coupled together for interactions over a communication network. Each of these aspects require a special-purpose simulation tool for correctly modeling and simulating its behavior. This presents a huge source of heterogeneity among the simulation models and tools.

[1] https://www.py4j.org

Many simulation integration platforms have emerged in the past that aim to provide tools, methods, and approaches to integrate the heterogeneous simulation models and tools in order to create a semantically correct integrated simulation. This system-of-systems simulation can support higher-level study of CPS that properly takes into account cross-domain interactions between its heterogeneous components. The main goal of simulation integration platforms is to enable integration of heterogeneous simulation models and tools in a logically and temporally coherent manner.

In this paper, we reviewed the core challenges of simulation integration and introduced a conceptual framework that describes three distinct horizontal integration platforms to integrate cross-domain interactions among CPS components, viz. the model-integration platform, the tool-integration platform, and the execution-integration platform. We also described how these three horizontal integration platforms are apparent in three widely used simulation integration platforms, viz. CPSWT, MOSAIK, and FMUs.

Further, we highlighted the need for interoperability among the different simulation integration platforms and presented our current work to enable interoperation among the above three integration platforms.

In the future, we plan to continue extending the inteoperability enabling tools and methods for interoperation among different simulation integration platforms. Another research direction is to look at formal specification of interoperation among simulation integration platforms to support more semantically correct and analyzable integrated simulations.

ACKNOWLEDGMENTS

This project at Vanderbilt University is supported in part by the Partnership for International Research and Education (PIRE) program and the Cyber-Physical Systems (CPS) program of NSF under award #1521617, and by NIST under award #70NANB18H269. The work at OFFIS and University of Oldenburg is supported in part by the German Research Council (DFG) as part of the project PIRE Science of Design of Societal Scale CPS, grant #LE 3131/7-1).

REFERENCES

[1] 2010. IEEE Std 1516-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Framework and Rules.
[2] 2019. FMI-standard. https://fmi-standard.org/
[3] 2019. Matlab/Simulink. https://www.mathworks.com/products/simulink.html
[4] 2019. Portico - RTI, URL: https://github.com/openlvc/portico.
[5] Y. Barve, H. Neema, S. Rees, and J. Sztipanovits. 2018. Towards a Design Studio for Collaborative Modeling and Co-Simulations of Mixed Electrical Energy Systems. In *2018 IEEE Int. Science of Smart City Operations and Platforms Engineering in Partnership with Global City Teams Challenge (SCOPE-GCTC)*. IEEE, 24–29.
[6] T. Blochwitz et al. 2012. Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International MODELICA Conference; September 3-5; 2012; Munich; Germany*. Linköping University Electronic Press, 173–184.
[7] Bradley Potteiger, William Emfinger, and Xenofon Koutsoukos. 2016. Evaluating the Effects of Cyber-Attacks on CPS using a HIL Simulation Testbed. *Resilience Week* (Aug. 2016).
[8] Dag Brück, Hilding Elmqvist, Sven Erik Mattsson, and Hans Olsson. 2002. Dymola for multi-engineering modeling and simulation. In *Proceedings of modelica*, Vol. 2002. Citeseer.
[9] Martin Burns, Thomas Roth, Edward Griffor, Paul Boynton, Janos Sztipanovits, and Himanshu Neema. 2018. Universal CPS Environment for Federation (UCEF). In *2018 Winter Simulation Innovation Workshop*.
[10] M. Büscher et al. 2014. Integrated Smart Grid simulations for generic automation architectures with RT-LAB and mosaik. In *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*. IEEE, 194–199.
[11] Gustavo Carneiro. 2010. NS-3: Network simulator 3. In *UTM Lab Meeting April*, Vol. 20.
[12] Xinjie Chang. 1999. Network simulations with OPNET. In *Proceedings of the 31st conference on Winter simulation: Simulation—a bridge to the future-Volume 1*. ACM, 307–314.
[13] Shigang Chen and Klara Nahrstedt. 1998. An overview of quality of service routing for next-generation high-speed networks: problems and solutions. *IEEE network* 12, 6 (1998), 64–79.
[14] Graham Hemingway, Himanshu Neema, Harmon Nine, Janos Sztipanovits, and Gabor Karsai. 2012. Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach. *Simulation* 88, 2 (2012), 217–232.
[15] Himanshu Neema. 2018. Large-Scale Integration of Heterogeneous Simulations. *Ph.D. Dissertation, Vanderbilt University* (Jan. 2018).
[16] Ethan K Jackson, Dirk Seifert, Markus Dahlweid, Thomas Santen, Nikolaj Bjørner, and Wolfram Schulte. 2009. Specifying and composing non-functional requirements in model-based development. In *International Conference on Software Composition*. Springer, 72–89.
[17] Janos Sztipanovits. 2007. Composition of Cyber-Physical Systems. *in Proc. of the 14th Annual IEEE Int. Conf. & Workshops on the Engineering of Computer-Based Systems (ECBS'2007)* (2007), 3–6.
[18] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. 2007. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer* 9, 3-4 (2007), 213–254.
[19] X. Koutsoukos et al. 2018. SURE: A modeling and simulation integration platform for evaluation of secure and resilient cyber–physical systems. *Proc. IEEE* 106, 1 (2018), 93–112.
[20] Robert H Lasseter and Paolo Paigi. 2004. Microgrid: A conceptual solution. In *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, Vol. 6. IEEE, 4285–4290.
[21] Miklós Maróti, Róbert Kereskényi, Tamás Kecskés, Péter Völgyesi, and Akos Lédeczi. 2014. Online collaborative environment for designing complex computational systems. *Procedia Computer Science* 29 (2014), 2432–2441.
[22] H. Neema et al. 2014. Design Space Exploration and Manipulation for Cyber Physical Systems. In *IFIP First International Workshop on Design Space Exploration of Cyber-Physical Systems (IDEAL'2014)*. Citeseer.
[23] H. Neema et al. 2018. Integrated Simulation Testbed for Security and Resilience of CPS. *The 33rd ACM/SIGAPP Symposium On Applied Computing* (Apr. 2018).
[24] Himanshu Neema, Jesse Gohl, Zsolt Lattmann, Janos Sztipanovits, Gabor Karsai, Sandeep Neema, Ted Bapty, John Batteh, Hubertus Tummescheit, and Chandraseka Sureshkumar. 2014. Model-based integration platform for FMI co-simulation and heterogeneous simulations of cyber-physical systems. In *Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden*. 235–245.
[25] H. Neema, G. Karsai, and A. H. Levis. 2015. *Next-Generation Command and Control Wind Tunnel for Courses of Action Simulation*. Technical Report no. ISIS-15-119. Institute for Software-Integrated Systems, Vanderbilt University.
[26] Neema, H., H. Nine, G. Hemingway, J. Sztipanovits, and G. Karsai. 2009. Rapid Synthesis of Multi-Model Simulations for Computational Experiments in C2. *Armed Forces Communications and Electronics Association - GMU Symposium, Critical Issues in C4I* (May 2009).
[27] Neema, H., J. Sztipanovits, M. Burns, and E. Griffor. 2016. C2WT-TE: A Model-Based Open Platform for Integrated Simulations of Transactive Smart Grids. *Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)* (Apr. 2016).
[28] Sebastian Rohjans, Edmund Widl, Wolfgang Müller, Steffen Schütte, and Sebastian Lehnhoff. 2014. Gekoppelte simulation komplexer energiesysteme mittels mosaik und FMI. *at-Automatisierungstechnik* 62, 5 (2014), 325–336.
[29] J. Sztipanovits, T Bapty, S. Neema, L Howard, and E Jackson. 2014. OpenMETA: A Model and Component-Based Design Tool Chain for Cyber-Physical Systems. In *From Programs to Systems – The Systems Perspective in Computing (FPS 2014)*. Springer, Springer, Grenoble, France.
[30] Janos Sztipanovits and Gabor Karsai. 1997. Model-Integrated Computing. *Computer* 30, 4 (1997), 110–111.
[31] A Varga. 2001. The OMNeT++ Discrete Event Simulation System. *In: Proceedings of the European Simulation Multiconference (ESM'2001)* (2001).
[32] Edmund Widl, Wolfgang Müller, Atiyah Elsheikh, Matthias Hörtenhuber, and Peter Palensky. 2013. The FMI++ library: A high-level utility package for FMI for model exchange. In *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2013 Workshop on*. IEEE, 1–6.
[33] Yuan Xue, T Busch, Michael Gacek, H Neema, G Karsai, and J Sztipanovits. 2010. A model-based integration of network emulation with HLA-based heterogeneous simulation environments. *ISIS* (2010), 10–107.