

PowerEnergy2017-3589

CYBER-PHYSICAL SYSTEM DEVELOPMENT ENVIRONMENT FOR ENERGY APPLICATIONS

**Thomas Roth
Eugene Song
Martin Burns**

Smart Grid & Cyber-Physical Systems Program Office
Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8200 USA
{thomas.roth,eugene.song,martin.burns}@nist.gov

**Himanshu Neema
William Emfinger
Janos Sztipanovits**

Institute for Software-Integrated Systems
Vanderbilt University
Nashville, TN 37212 USA
{himanshu,emfinger,sztipaj}@isis.vanderbilt.edu

ABSTRACT

Cyber-physical systems (CPS) are smart systems that include engineered interacting networks of physical and computational components. The tight integration of a wide range of heterogeneous components enables new functionality and quality of life improvements in critical infrastructures such as smart cities, intelligent buildings, and smart energy systems. One approach to study CPS uses both simulations and hardware-in-the-loop (HIL) to test the physical dynamics of hardware in a controlled environment. However, because CPS experiment design may involve domain experts from multiple disciplines who use different simulation tool suites, it can be a challenge to integrate the heterogeneous simulation languages and hardware interfaces into a single experiment. The National Institute of Standards and Technology (NIST) is working on the development of a universal CPS environment for federation (UCEF) that can be used to design and run experiments that incorporate heterogeneous physical and computational resources over a wide geographic area. This development environment uses the High Level Architecture (HLA), which the Department of Defense has advocated for co-simulation in the field of distributed simulations, to enable communication between hardware and different simulation languages such as Simulink and LabVIEW. This paper provides an overview of UCEF and motivates how the environment could be used to develop energy experiments using an illustrative example of an emulated heat pump system.

Introduction

A cyber-physical system (CPS) consists of a set of interacting cyber-physical devices where each device contains some cyber computation that can sense events from and actuate changes on a physical infrastructure. Examples of CPS include smart cities, intelligent buildings, and the smart grid. One method to validate a CPS design uses hardware-in-the-loop (HIL) in conjunction with simulations to test the runtime dynamics of a cyber-physical device in a virtual test environment. A challenge of experiments that incorporate both HIL and simulations is that they often require a testbed that integrates hardware components with multiple, heterogeneous simulation environments.

A large number of HIL testbeds which offer unique experimental opportunities cannot be replicated due to limitations in both hardware cost and development time [1–5]. These testbeds often have different architectures and utilize different simulation languages because of their independent development histories, and an experiment tailored for one testbed might not be compat-

Official contribution of the National Institute of Standards and Technology; not subject to copyright in the United States. Certain commercial products are identified in order to adequately specify the procedure; this does not imply endorsement or recommendation by NIST, nor does it imply that such products are necessarily the best available for the purpose.

ible with another architecture. The inability to exploit the full range of available resources in the CPS landscape leads to segregated groups of researchers who are experts in a single testbed environment but face challenges in the adoption of external research advances. In addition, integrated experiments for CPS require access to resources pooled from multiple domains to produce faithful models of the deployed system. For example, experiments on smart cities may involve collaboration across domains such as transportation, energy, and emergency response. An experiment should integrate models developed in those domains, which may involve domain-specific tools (e.g. a traffic simulator written in C++), to achieve the most realistic result.

NIST envisions a universal CPS environment for federation (UCEF) which enables experiments to exploit multiple testbed architectures using a common interface. The United States Department of Defense mandated a common integration platform in the field of distributed simulators called the High Level Architecture (HLA) [6]. This paper demonstrates the use of HLA in the design and implementation of cyber-physical devices using an integration architecture that supports collaboration between physical hardware and simulations. The approach is highlighted using an example CPS implementation of an HVAC system controlled by a thermostat with a remote temperature sensor, which is a straightforward and well understood application that does not require deep domain expertise to comprehend.

The rest of the paper is organized as follows. Section II provides an overview of HLA and the design process to implement an HLA federation. Section III demonstrates this design process in an example CPS through implementation of a distributed HVAC system. Section IV outlines other work on the integration of HLA with hardware, and the paper concludes with Section V.

High Level Architecture

HLA is an IEEE standard for distributed simulation in which individual simulations called federates join together to form a cooperative federation [6]. All federates in a federation interact using a Run-Time Infrastructure (RTI) software implementation of a set of HLA services such as publish-subscribe messaging, logical time management, and distributed object management. Data exchanges between the federates must adhere to a federation object model (FOM) which defines the set of messages understood by the federation. Although the original intent of HLA was to allow federated co-simulation of simulation platforms such as MATLAB and Modelica, a CPS federate could represent a cyber-physical device. This section provides a brief overview of this paper’s approach to designing an HLA federation with hardware-in-the-loop. The overview is based on a model-based simulation integration environment developed and maintained by Vanderbilt University called the Command and Control Wind Tunnel (C2WT) [7], but has been sufficiently generalized to be applicable to alternative HLA development environments.

Federation Stack Architecture

HLA does not mandate a specific RTI implementation, which can consist of two different types of components. A Local RTI Component (LRC) provides an Application Program Interface (API) to interface federates with the RTI, and a Central RTI Component (CRC) coordinates the other run-time components. A specific RTI implementation may provide a centralized CRC, multiple hierarchical CRCs, or no CRC. The results in this paper use an open-source RTI implementation called Portico which implements the LRC at each federate and requires no CRC [8]. Fig. 1 shows a federation stack architecture for this implementation that illustrates the necessary components for a federate. This figure contains three example federate types: a simulation, a cyber-physical device, and a federation manager that drives an experiment.

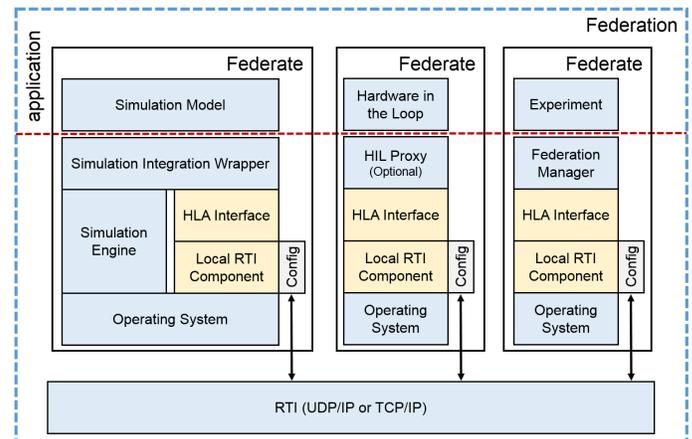


FIGURE 1. Federation Stack Architecture

Each federate has a *Local RTI Component* implementation which enables it to communicate with the federation, and all federates must use the same LRC implementation to ensure coherent communication between the federation members. The Portico LRC implementation uses either TCP/IP or UDP/IP sockets for its intra-federation communication. On top of this communication infrastructure, an *HLA Interface* exposes the set of standardized services available for federates. For the C2WT integration environment, the HLA interface is a Java abstract class which exposes the various HLA services as Java functions. The implementation of the LRC and its HLA interface are uniform across all of the federate types.

For simulation platforms such as MATLAB, the federate must also contain a *Simulation Engine* that runs the simulation models. The simulation engine may not have a native RTI interface. In order to make these platforms compatible with HLA, an adapter labeled the *Simulation Integration Wrapper* must be

developed to translate the HLA interface into the language of the simulation engine. These wrappers are reusable and all of the models for the same simulation engine use the same wrapper implementation to join an HLA federation.

For hardware-in-the-loop, there is no simulation engine and the ease of HLA integration depends on whether the hardware controller can run the RTI implementation and its dependencies. The Portico LRC, for example, is written in Java and might not be able to run on all embedded systems. For the cases that do not support Java, an *HIL proxy* for the hardware must be deployed on a separate, compatible processor. This proxy federate serves as an adapter between the hardware and the HLA interface and can use any protocol to connect to the hardware while using the LRC’s communication stack to expose the hardware interface to the federation.

Each federation has an additional *Federation Manager* that coordinates the other federates and runs the experiment. The federation manager’s responsibilities include creation of the federation and management of synchronization points such as simulation start. The federation manager can also affect the operational flow of an experiment based on its intermediate results, providing a powerful and flexible experiment design capability. This federation manager is not part of the HLA standard, but is a natural way to provide the ability to affect the runtime behavior of a federation execution for an experiment. It is important to note that the federation manager is a normal HLA federate and has the same access to the HLA services as all other federates in the federation.

Federate Development Workflow

The NIST vision of a UCEF testbed consists of an open database of simulation and hardware federates as described in the prior section that can be composed into complex cyber-physical experiments. One requirement for this vision is an integration environment where the various pieces of the stack architecture can be developed and assembled. Fig. 2 shows the development workflow to create the necessary parts for this architecture.

The Web Generic Modeling Environment (WebGME), developed by Vanderbilt University, provides tools and methods for creating meta-models for rich Domain-Specific Modeling Languages (DSMLs) which can then be used to design domain models and generatively synthesize programs [9]. WebGME has been leveraged to create a DSML for modeling HLA federations. In practice, WebGME could be replaced with an alternative tool in the development workflow as long as the new tool maintained the same interfaces shown in the figure. One function of WebGME is to model *Federate Interfaces*, which in the language of HLA would be the Simulation Object Model (SOM) which describes the publish-subscribe model for a federate. A second function of WebGME is to use code generation to transform the modeled federate interfaces into basic federate implementations,

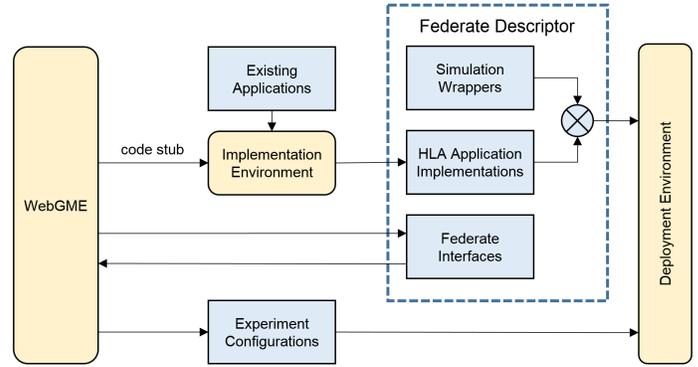


FIGURE 2. Federation Development Workflow

or *code stubs*. This generated code stub contains the HLA interface block shown in Fig. 1, which must then be extended into either a simulation or hardware application. An *Implementation Environment* (such as MATLAB or Java) must be used to either extend this code stub or integrate it with an *Existing Application* to produce an *HLA Application*. If the HLA application uses a simulation platform that requires custom wrapper code to integrate with HLA, then the application code must be combined with the *Simulation Integration Wrapper* prior to deployment. A federate descriptor combines the federate interface, its implementation in either simulation or hardware, and any simulation wrapper necessary to integrate the federate with HLA.

After a database of federate descriptors has been developed, a subset of the descriptors must be combined together to form a federation that performs some useful experiment. Another function of WebGME is the ability to produce *Experiment Configurations* which includes programming the federation manager’s runtime behavior using a graphical language such as courses of action [10] or colored Petri nets [11]. The experiment configuration defines which federate descriptors will participate in the federation, where each federate descriptor will be deployed, and what script the federation manager will use for the experiment.

Example HVAC System

This section introduces a simple heating, ventilating, and air conditioning (HVAC) system that will be used to demonstrate the implementation of a hardware federate in HLA. An HVAC system was selected as the example implementation because it is a well understood application.

Fig. 3 shows the federation design for a thermostat that controls the HVAC system using a remote temperature sensor. The federation consists of three federates excluding the federation manager. The HVAC federate controls a relay board connected to two fans and a Peltier heat pump to change the temperature of a pair of heatsinks. The remote temperature sensor federate reads the temperature measurement of a sensor placed inside one

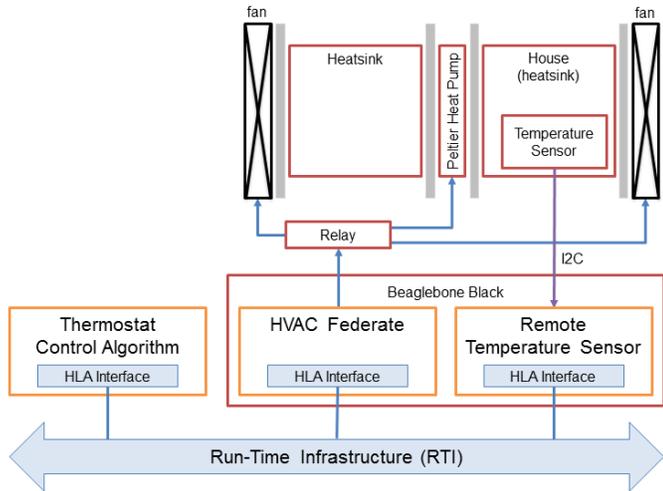


FIGURE 3. Federated HVAC System Design

heatsink that represents a house. Both of these federates run on a BeagleBone Black (BBB) and communicate with a thermostat federate over the RTI. The thermostat can remotely pull the house temperature from the temperature sensor via the BBB, compare it to some internal set point, and then remotely issue commands to the heat pump to maintain the set point.

Hardware Implementation

The thermostat issues heating and cooling commands to the heat pump to control the temperature of the heatsink that represents a house. A Peltier heat pump can transfer heat from one side of the device to the other based on the direction of the electric current. This enables the heat pump to provide both of the required heating and cooling functions. Fig. 4 shows the hardware implementation of the federated HVAC system. At the top of the figure, a 12 V, 5 A Peltier Thermo-Electric Cooler Module and Heatsink Assembly is mounted onto an additional heatsink and fan assembly with thermal paste and an aluminum clamp. A Microchip MCP9808 temperature sensor is inserted into the right heatsink to measure the temperature of the house. The temperature sensor communicates using a 2-wire inter-integrated circuit (I2C) bus that allows for serial communication between a master device (e.g. BBB) and its slave devices (e.g. temperature sensor).

The BBB is an embedded computer with an AM335x 1 GHz ARM Cortex-A8 processor that is compatible with Debian, Android, Ubuntu, and Cloud9 [12]. The libbulldog Java library for the BBB provides low-level IO capabilities for embedded linux systems which includes the ability to access the general purpose IO pins and communicate to slave devices using I2C [13]. Fig. 5 shows how the BBB was integrated with the hardware in the HVAC system. A 4-channel relay board was inserted between the BBB and the heatsinks and fans, with one pin allocated for

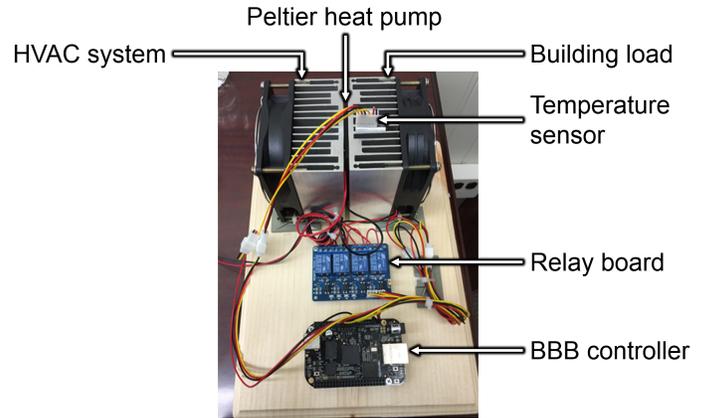


FIGURE 4. HVAC System Hardware Implementation

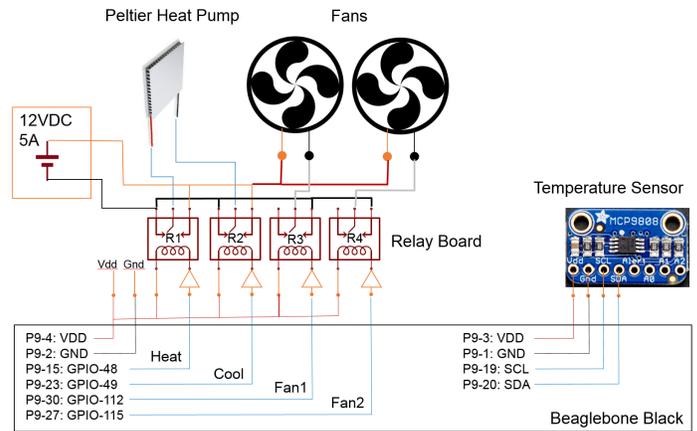


FIGURE 5. BeagleBone Black Pin Connections

each fan and two pins for the heat pump. The fan signal represents its on/off state, while the two heat pump signals determine whether the unit is off, heating, or cooling. The BBB is also connected to the temperature sensor using its I2C serial communication ports. These connections allow the federates running on the BBB to read the temperature measurement and control the heat pump and associated fans.

Federate Implementation

The hardware implementation discussed so far falls into the *Existing Applications* box in Fig. 2. This section will describe how to create the three required federate descriptors to turn the hardware into an HLA application.

The first step is to design the various federate interfaces in WebGME. Fig. 6 shows the federation design for the HVAC system in this environment. The federation consists of three federates: a hardware HVAC federate which controls the relay board, a HIL proxy federate for the temperature sensor, and a soft-

ware thermostat federate for the control algorithm that uses the temperature measurement to generate heating and cooling commands. The temperature sensor federate publishes a single interaction called `SensorMessage` which contains the temperature measurement from the sensor located in the house heatsink. The thermostat federate subscribes to this interaction, with the expectation that the temperature measurement will be used to perform some amount of internal computation. The result of this computation is published to the HVAC federate through the `ControlMessage` interaction which contains four bit values for the relay board in the hardware implementation.

The code generation capability of WebGME can generate code stubs from these federate interfaces which contain boilerplate HLA code for connecting to the Portico LRC implementation. For the thermostat control algorithm, this code stub was extended into a simple Java program that extracts the temperature measurement from the `SensorMessage`, compares the measurement to an internal set point, and generates heating or cooling commands to maintain that set point. The HVAC federate was also extended into a Java program that turns the generated heating and cooling commands into electronic actuation. Its implementation leverages the `libbulldog` library which provides a Java interface to the BBB general I/O pins connected to the relay board. The temperature sensor required an HIL proxy federate to translate between the communication infrastructure provided by the Portico LRC implementation and its I2C serial bus. This proxy federate uses I2C to read the measurement produced by the temperature sensor, package the measurement in an HLA message format, and broadcast the measurement to the federation. The temperature sensor proxy federate was also developed in Java and used the same `libbulldog` library to create an I2C connection between the BBB and the temperature sensor device. Fig. 7 shows a sequence diagram of one round of message exchanges between the three federates during the federation execution.

The complete federate descriptor for each federate consists of its Java implementation and the WebGME project that defines its interface. Because the HVAC system did not require a simulator, none of the federate descriptors include a simulation integration wrapper. Now an experiment must be designed that describes the configuration in which the federate descriptors will be used in a larger federation.

Experimental Federate Design and Results

The experiment was configured to run both the HVAC and temperature sensor federates on the BBB and the thermostat federate on an Ubuntu desktop machine. The Ubuntu machine was connected to the BBB using a USB cable, and all communication used a USB network interface. The thermostat federate was configured to maintain a temperature set point of 38°C with a tolerance of 1°C. As mentioned before, an additional federation man-

ager federate was used to create the federation and coordinate the progression of the experiment. The federation manager was deployed on the Ubuntu machine using the experimental configuration generated by WebGME to inform it of the other federates expected to participate in the federation. Once all of the federates had joined the federation execution, the federation manager began a simulation that was configured to run for 180 seconds. Fig. 8 shows the temperature measurement read by the BBB during the federation execution. The BBB federate that controls the heatsink temperature was implemented using a simple control algorithm that did not model the physical system dynamics. Using the tools described in this paper, this control federate could be replaced with a federate using a different control algorithm that includes a prediction model to keep the temperature within the 1°C tolerance. The HLA implementation of the HVAC system with a remote temperature sensor functions as intended, and the federates coordinate over HLA to maintain the desired temperature set point.

Related Work

The original intent of HLA was to enable co-simulation between simulators with no notion of the hardware-in-the-loop described in this paper. There are pure simulation environments that integrate multiple tools using HLA federation to model more complex systems appropriate for the CPS domain. EPOCHS is one example which uses HLA to combine power simulation software with a network simulator to enable more accurate simulation of smart grids [14]. These simulation environments are useful experimental tools, but are not readily applicable to HIL testbeds. There have been several efforts which have extended HLA to support hardware for specific experiments in the domains of mechatronic systems [15], power systems [16], and embedded devices [17]. These efforts were focused on domain specific experimental outcomes rather than documentation of the development process for hardware federate interfaces, and their approaches are consistent with the architecture detailed in this paper.

In addition to efforts to develop HLA interfaces for specific simulations or hardware, several development environments to design these interfaces have emerged in recent literature. The authors in [18] extend the SysML modeling language to support HLA federation design and add a code generation capability to transform SysML models into code stubs for federate implementation. The authors in [19] introduce the architecture and workflow required to integrate new simulators with HLA using a similar SysML approach. Both of these environments are consistent with the figures and discussion in Section II with the exception of modeling using SysML instead of the model integration language for WebGME.

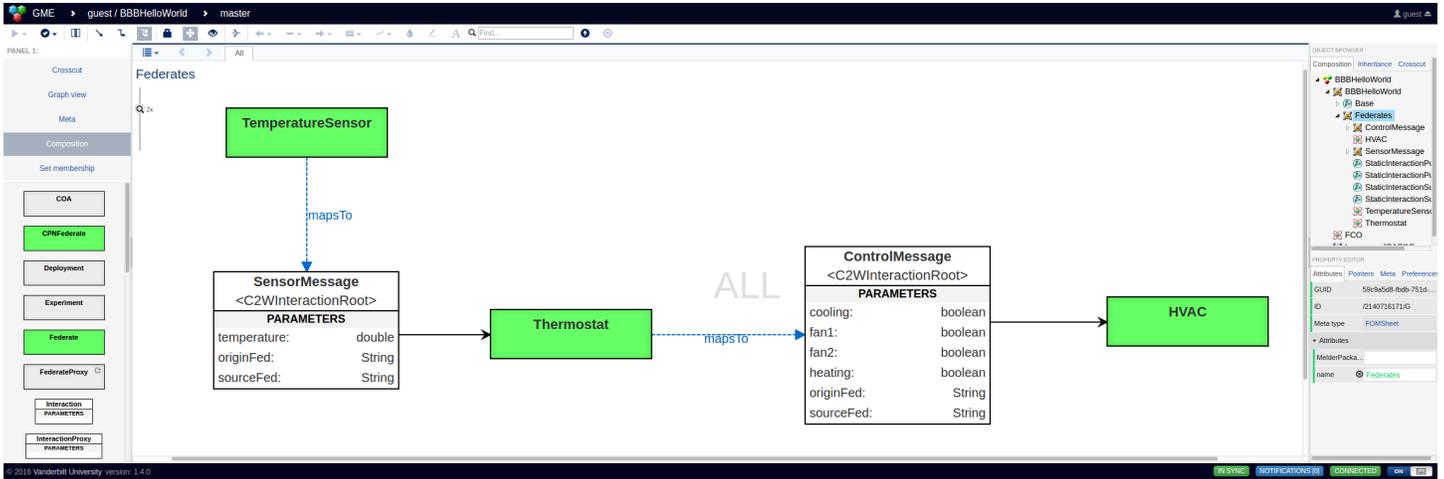


FIGURE 6. Federation Design of the HVAC System in WebGME

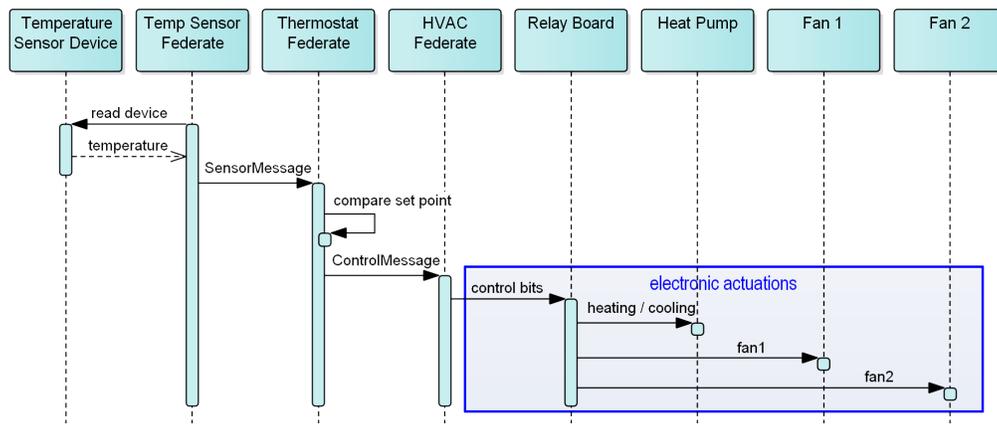


FIGURE 7. Sequence Diagram of the Federation Execution

Conclusion

This paper demonstrated how to design and implement HLA federates for cyber-physical devices using an example HVAC application. The original intent of HLA was to enable co-simulation of discrete simulation platforms, but the design and implementation of simple hardware federates can easily conform to the standard. The tools and design process described in this paper can be used to integrate more complex hardware implementations with HLA, leading to the development of federates that encapsulate entire testbed architectures.

Further efforts to integrate complex hardware with HLA could potentially enable a public database of federate descriptors which includes both simulation and hardware from which an experiment could be assembled to test sophisticated energy applications. However, adherence to the HLA standard does not require documentation of a federate’s functionality beyond the simulation object model that describes data exchanges in terms

of publications and subscriptions. For a CPS, the data exchange model is insufficient to capture the complexities of a physical system which might have instability concerns related to the timing and values of the messages it receives from different sources. One aspect missing from the presented architecture is a means to document these interoperability constraints which would enable a domain expert to determine when a set of federate descriptors were composable. Future work will begin to enumerate the interoperability constraints that must be documented for each federate.

The goal of this research is to realize federate descriptors for testbeds and refine the federation development workflow to support multiple testbed architectures. The end result will be the design of a universal CPS environment for federation (UCEF) that is flexible enough to run experiments across multiple domains using resources from multiple HIL testbeds. NIST is working with Vanderbilt University and others to develop the components

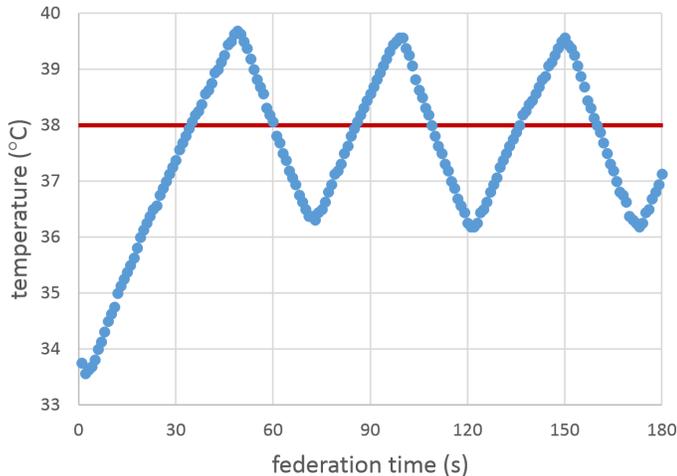


FIGURE 8. Temperature during Federation Execution

of this design and package them into a development environment to readily model, implement, and deploy CPS experiments.

REFERENCES

- [1] Chen, X., and Sun, J., 2011. "Characterization of inverter-grid interactions using a hardware-in-the-loop system testbed". In 2011 8th International Conference on Power Electronics and ECCE Asia (ICPE & ECCE), IEEE, pp. 2180–2187.
- [2] Prokhorov, A. V., Gusev, A. S., and Borovikov, Y. S., 2013. "Hardware-in-the-loop testbed based on hybrid real time simulator". In Innovative Smart Grid Technologies Europe (ISGT EUROPE), 2013 4th IEEE/PES, IEEE, pp. 1–5.
- [3] Stanovich, M. J., Leonard, I., Sanjeev, K., Steurer, M., Roth, T. P., Jackson, S., and Bruce, M., 2013. "Development of a smart-grid cyber-physical systems testbed". In Innovative Smart Grid Technologies (ISGT), 2013 IEEE PES, IEEE, pp. 1–6.
- [4] Pang, X., Wetter, M., Bhattacharya, P., and Haves, P., 2012. "A framework for simulation-based real-time whole building performance assessment". *Building and Environment*, **54**, pp. 100–108.
- [5] Haves, P., and Xu, P., 2007. "The building controls virtual test bed a simulation environment for developing and testing control algorithms, strategies and systems". In Proc. of the 10-th IBPSA Conference, pp. 1440–1446.
- [6] , 2010. "IEEE standard for modeling and simulation (M&S) high level architecture (HLA)– framework and rules". *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, Aug, pp. 1–38.
- [7] Hemingway, G., Neema, H., Nine, H., Sztipanovits, J., and Karsai, G., 2012. "Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach". *Simulation*, **88**(2), pp. 217–232.
- [8] Portico. <https://github.com/openlvc/portico>. [accessed 11-March-2016].
- [9] Maróti, M., Kecskés, T., Kereskényi, R., Broll, B., Völgyesi, P., Jurácz, L., Levendovszky, T., and Lédeczi, Á., 2014. "Next generation (meta) modeling: Web-and cloud-based collaborative tool infrastructure.". In MPM@ MoD-ELS, pp. 41–60.
- [10] Neema, H., Karsai, G., and Levis, A., 2015. "Next-generation command and control wind tunnel for courses of action simulation". *Technical Report ISIS-15-119*, May, p. 119.
- [11] Jensen, K., 1987. "Coloured petri nets". In *Petri nets: central models and their properties*. Springer, pp. 248–299.
- [12] Beaglebone black. <https://beagleboard.org/black>. [accessed 8-April-2016].
- [13] libbulldog. <http://beagleboard.org/project/libbulldog>. [accessed 02-June-2016].
- [14] Hopkinson, K., Wang, X., Giovanini, R., Thorp, J., Birman, K., and Coury, D., 2006. "EPOCHS: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components". *IEEE Transactions on Power Systems*, **21**(2), pp. 548–558.
- [15] Kanai, S., Miyashita, T., and Tada, T., 2007. "A multi-disciplinary distributed simulation environment for mechatronic system design enabling hardware-in-the-loop simulation based on HLA". *International Journal on Interactive Design and Manufacturing (IJIDeM)*, **1**(3), pp. 175–179.
- [16] Jablkowski, B., Spinczyk, O., Kuech, M., and Rehtanz, C., 2014. "A hardware-in-the-loop co-simulation architecture for power system applications in virtual execution environments". In 2014 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), IEEE, pp. 1–6.
- [17] Brito, A. V., and Nascimento, T. P., 2015. "Verification of embedded system designs through hardware-software co-simulation". *International Journal of Information and Electronics Engineering*, **5**(1), p. 68.
- [18] Bocciarelli, P., D'Ambrogio, A., Giglio, A., and Gianni, D., 2013. "A SaaS-based automated framework to build and execute distributed simulations from SysML models". In 2013 Winter Simulation Conference (WSC), IEEE, pp. 1371–1382.
- [19] Jain, A., Fujimoto, R., Crittenden, J., Liu, M., Kim, J., and Lu, Z., 2015. "Towards automating the development of federated distributed simulations for modeling sustainable urban infrastructures". In 2015 Winter Simulation Conference (WSC), IEEE, pp. 2668–2679.