# FORMALIZING CYBER-PHYSICAL SYSTEM INTERFACES USING DEVS

Rishabh Sudhir Jiresal
Gabriel Wainer

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Dr, Ottawa, ON, CANADA
rishabhjiresal@cmail.carleton.ca, gwainer@sce.carleton.ca

## ABSTRACT

Cyber-Physical Systems (CPS) integrate embedded computing into the physical environment. CPS are composed of tight coupling of the physical and cyber worlds, and the interfaces are interactions that are defined as a bridge between these worlds. We introduce an architecture of the CPS interface using Discrete EVent System Specifications (DEVS) that mimics the functionality of CPS interactions. We propose a DEVS model that portrays the complete working of the CPS interface, and we also implement and evaluate this interface on real-time hardware. The architecture is tested by building a synthetic environment that includes multiple test cases in the simulation as well as real-time implementation. Additionally, we apply this framework to a practical case study in the field of building information modeling.

**Keywords:** Cyber-Physical Systems, RT-DEVS, Discrete Event Modelling, DEVS.

## 1 INTRODUCTION

Cyber-physical systems (CPS) are next-generation engineered systems that integrate embedded computing into the physical environment (Marwedel 2006) (Gill, 2008). They can be seen as embedded systems with tightly coupled computational and physical components. Some examples of the application of CPS include air traffic control systems, medical devices, and equipment, railway cross control systems, automotive airbag, smart homes, building systems, etc. (Khaitan and McCalley 2015).

CPSs are regarded as "safety-critical" in many applications (Fromel 2016), as improper interactions in the CPS could lead to life loss, and health or environmental hazards. The dependability of a CPS relies on the coordination of interactions in the system; for instance, the data gathered from sensors influence the decision-making related to its control actions (Kaur 2018). Modern CPS are complex, and we need to evaluate their performance under various conditions before they can be implemented, and Modeling and Simulation (M&S) is an effective method to verify the design and implementation details of such CPS.

CPSs can be seen as a combination of the physical and cyber worlds with interfaces between them and their interactions (Gunes 2014) (Fromel 2016). Physical interfaces use devices (sensors, actuators, control) to interact with the real world by measuring and manipulating physical variables (temperature, pressure, motion), and then transfer their values to computational components. Cyber interfaces refer to components for communication and control between the physical and the computational components and facilitate the flow of data between them, allowing for real-time monitoring, analysis, and control of physical processes. These physical and cyber interfaces are essential components of CPS, as they enable integration, data acquisition, control, and adaptability required for these systems to operate effectively.

*ANNSIM '23, May 23-26, 2023, Mohawk College, ON, CANADA;* 

Both physical and cyber interactions are important components of CPS, and although there is a variety of research on the development of sensors, actuators, and embedded computational systems, there is little research on interfaces. Considering these issues, the objective of this work is to investigate the definition of interfaces for M&S, including a formal definition, an architecture, and a workflow for designing CPS interfaces. We will show how behavioral issues in the CPS can be mitigated if the system interactions/interfaces are formally defined. We use Discrete Event System Specifications (DEVS) and consider the CPS interface interactions using such specifications (Zeigler 1984).

The result is the definition of DCIF (DEVS CPS Interface Framework), a discrete-event model specification that defines the working of the CPS interface and allows both simulation and real-time execution. As it will be seen later, DEVS is well-fitted to define the DCIF, which defines core functionality of CPS interfaces to portray the interactions in the system. DCIF in DEVS allows an efficient implementation and includes a sensor fusion framework (Boi-Ukeme and Wainer, 2020). The introduction of DCIF contributes towards enhancing the measured data reliability through sensor fusion, data storage and analysis. We show the use of DCIF through a case study based on a prototype application of a BIM model of Carleton University where we use DCIF for data acquisition and storage and Building Information Models (BIM).

## 2 CYBER-PHYSICAL SYSTEMS (CPS)

CPS are computer systems that bridge the cyber-world of computing and communications with the physical world (Rajkumar, et al. 2010). According to (Khaitan and McCalley 2015), current research in cyber-physical systems can be classified into three categories: Design, Aspects/Issues, and Applications. Cyber-physical systems are usually "systems of systems" where complex and heterogeneous systems interact. Engineered systems are used in many societally critical application domains (medical, transportation, and building control) that need to be secure, dependable, and robust. They use real-time embedded systems for distributed sensing, computing, and control over communication networks, as well as multi-objective optimization, high-level decision-making algorithms, and formal verification (Kim and Kumar 2012).

CPSs are typically expected to be always available, display appropriate behavior, and survive failures, including sensor reliability (Castaño, et al. 2019). Within a CPS, the data gathered from the sensors influences the decision-making related to its control actions. The resulting decisions are materialized using actuators and thus the loop between the physical and cyber components is closed. Given the importance of the data at this level, it must be accurate, accessible, timely, and consistent (Sanislav, et al. 2019).

CPS can be built and tested using Hardware-In-the Loop Simulation (HIL), a type of real-time simulation that can be used to test a control system on the intended target hardware by creating a virtual real-time environment that replicates the physical system to be controlled (Bin Lu 2007). Numerous applications of HILS exist; (A. M. Astorga 2011) used LabVIEW to test control algorithms for an autonomous marine vehicle; (Irwanto and Artono 2018) proposed a flying vehicle HILS to improve cost, and time in several procedures for the flight tests; (Pratt, et al. 2017) used it for Air Conditioner predictive control; (Barreras, et al. 2016) for a battery management system. Systems design using HILS include all the parts of the system (Carrijo, Oliva and Leite Filho 2002), a computer to for control, and simulation for the rest of the system. Once the simulation is successful, the hardware interfaces are defined (Köhler 2011).

CPS normally have real-time constraints to be met, but traditional real-time embedded software development methods have limitations because they use different tools and environments for each process step. In (Hong, et al. 1997), the authors introduced RT-DEVS, a formal definition for the design of embedded hardware and software using a model-based approach which is an extension of DEVS, Discrete Event System Specification (Zeigler 1984). In (Wainer 2015) we introduced a model-driven framework to develop CPS based on DEVS and RT-DEVS that brought in a reliable way to incrementally develop embedded

applications with HILS, which was to build different applications, ranging from a control system for a quadcopter (Ruiz-Martin, et al. 2019), building control systems (Earle, Bjornson and Boi-Ukeme, et al. 2019) for lighting and emergency systems based on occupancy, and others. Different tools like PowerDEVS (Bergero and Kofman 2010) have been used to build CPS applications using DEVS models in C++ that can then be coupled graphically in hierarchical block diagrams to create complex systems, similarly to Real-Time Cadmium (RT-Cadmium) (Earle, Bjornson and Ruiz-Martin, et al. 2020).

The domain of application of this research is the design of control systems in buildings, including 3D visualization. We use Building Information Modeling (BIM), which facilitates the representation of architectural elements in terms of their geometric and functional properties and interactions (Ghaffarianhoseini, et al. 2017) in 3D. BIM helps in solving functional, informational, and organizational issues, as well as technical issues: (Volk et al 2014). Our long-term goals include building a framework to experiment with Fault Detection and Diagnosis (FDD) in BIM, which requires a knowledge base to obtain the status of the physical aspects of a building in order to build simulation models (Katipamula and Brambley 2005). FDD systems should detect faults and diagnose their causes (along with correction before additional damage or loss of service) and can be seen as consisting of three key processes: fault detection, identification, and isolation. (Boi Ukeme et al. 2020) presented a scheme to detect and diagnose these faults based on model-driven FDD using DEVS. The method is based on sensor fusion, which involves association, correlation, estimation, and combination of data from several sources (Khaleghi, et al. 2013), which has advantages for data authenticity and availability (D.L. Hall 1997). In this area, (Motawa and Almarshad 2013) developed a system to capture information and knowledge of building maintenance operations and (Lee, Cha and Park 2016) built a data acquisition system and a BIM to monitor building's energy consumption.

Our research uses the results above as a base for the definition and implementation of a formalization of CPS interfaces in DEVS, combining RT-DEVS, sensor fusion, and HILS, and using BIM to provide visualization and to build case studies in the field of BIM and building maintenance.

## 3 DEVS CPS INTERFACE FRAMEWORK (DCIF)

As we discussed earlier, the interactions between the cyber and physical worlds in a CPS are done through interfaces. There are two types of interactions, physical and message-based. The electrical signals can be seen in the cyber world as "messages" interchanged between components, known as message-based interactions. These messages, which are passed through the CPS interface, normally follow an organization and a set of rules. If we formalize/standardize the behavior of the CPS interface in the system, we can make the behavior of the cyber-physical system more predictable, manageable, and consistent.

Figure 1 shows a high-level description of a CPS with both worlds connected through an interface. As shown in the figure, there is an exchange of information between the two spaces of the CPS through the interface. The control system receives the data from the sensors at a specific time interval and once the data is received, it decides on the actuation and sends a command to the actuator hardware. The transmission and reception of the data and the commands are done through the CPS interface framework.

For the formalization of the CPS interface, there are important design considerations: (a) the formalization does not consider any communication protocol as its part, for the sensors or actuator connections (the formalization of the interface is hardware independent); (b) initialization time for the sensors is not considered in the design; (c) the framework is time-synchronous (asynchronous inputs based on interrupts are not taken into consideration in this iteration; support for these inputs can be provided by adding some additional handlers).

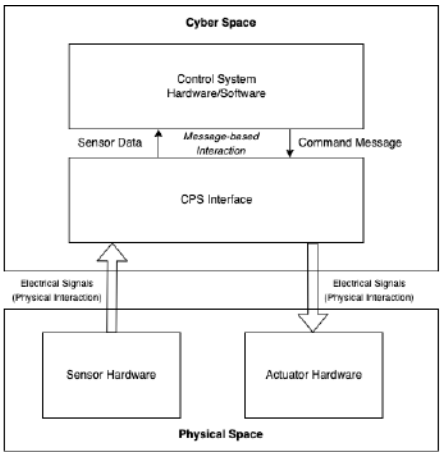


Figure 1: High-Level Design of Cyber-Physical System.

There are tasks in the framework that need to occur before sending out any data to/from the control system. To organize these tasks, the CPS interface is made up of components (layers) responsible for specific tasks. We consider three functional layers in a CPS interface: the Cyber-Physical Layer, the Informational Layer, and the Service Layer (Figure 2). The layers are defined using DEVS.

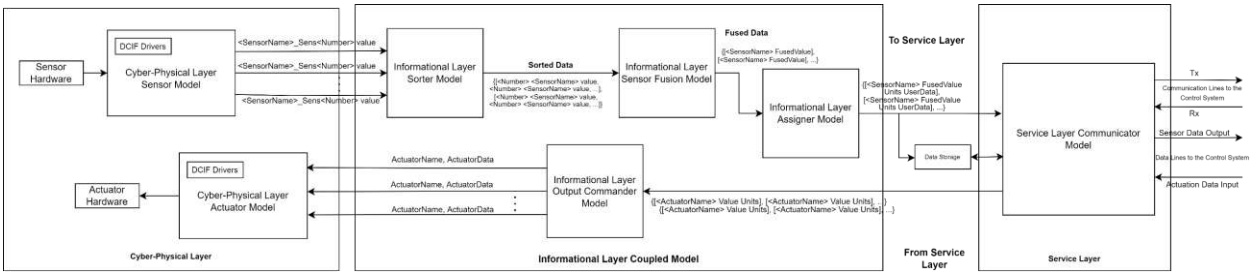


Figure 2: Cyber-Physical System Interface Framework (DCIF) Top Model.

There is two-way communication between the layers. The message-based interactions involve data correction, manipulation, and processing which is conducted in the DEVS models in each layer. These messages between the models have a unique structure for each model. We also propose a message structure to pass information between the layers. These messages play an important part in the CPS interface framework and each model in the DCIF has a particular message structure for the inputs and the outputs.

### 3.1 The Cyber-Physical Layer

At the Cyber-Physical Layer, the information is represented by bit patterns and is exchanged between this layer and the Informational Layer. The bit pattern is obtained by using the electrical signals and finding their state at a particular time. The sensors can be connected to the processing hardware using any communication protocol (SPI, I2C, UART) but the interface framework is independent of the communication method or the hardware used. The Cyber-Physical Layer gets the data from the sensors as electrical signals and converts them into a bit pattern and further to a message that can be transferred between to the Informational Layer. The Cyber-Physical Layer also receives actuation commands from the control system, converts the data into bit patterns and sends it out to the required actuator. The Cyber-Physical Layer includes important properties such as bit rates, energy levels, transmission medium, if they are signals or bit patterns (representation of the data according to the communication protocol with the sensors/actuators), etc. It includes all these "lower-level details" defined for the CPS to run on specific hardware.

This Layer is formalized as a composite of two DEVS models: Sensor and Actuator (as well as the DCIF device drivers responsible for communicating with the hardware). Sensors are connected to the DCIF through the Sensor Model, which manages the connection to the hardware and fetches data using the DCIF drivers. The Actuator receives commands and it handles the actuation of the hardware using the DCIF drivers. This Layer access hardware functions without knowing specific details about the hardware being used. The DCIF drivers are a unified wrapped library built by multiple hardware manufacturers according to the type of sensor/actuator used (which provides similar functionality with varied working principles and precision). The DCIF drivers library enables us to switch Sensor models with no impact on the rest of the system. The drivers return values with specific standardized units for each sensor family which makes it comparable with any other similar sensors. These specific standardized SI units are retained throughout the DCIF and are later used in the informational layer to make the data presentable.

The model in the layer uses its own message structure. For instance, the message from Actuator consists of two parts: `TIMESTAMP <ActuatorName><Number> value`

A general description of the formal specification of the Actuator DEVS atomic model is as follows:

$$AM = \langle S, X, Y, \delta int, \delta ext, \delta con, \lambda, ta \rangle$$

```
S = {sendingDataToActuator ∈ {true, false}}
X = {actuator_message}
Y = {} // outputs to the hardware are accessed through
                        the DCIF drivers
δext (s,e,x) = {
    if sendingDataToActuator = false, then
        actuator_name = actuator_message.name
        actuator_value = actuator_message.value
        sendingDataToActuator = true
}
δcon = δint; δext;
```

```
δint (s) = {
    if sendingDataToActuator = true, then
        sendingDataToActuator = false
}

λ (sendingDataToActuator=true) = {
   Call DCIF drivers with the actuator values as the
     parameters. Apply signal conditioning if applicable
}

ta (sendingDataToActuator = true) = actuationTime
ta (sendingDataToActuator = false) = INFINITY
```

According to the formal definition, there are no outputs (Y) as the actual outputs is sent to the actuator hardware connected to the model, accessed through the DCIF drivers called in the output function (λ). The DCIF drivers initialize the communication of the controller with the actuators using the lower-level hardware device drivers on model instantiation. The controller uses the DCIF drivers to actuate the hardware according to the values passed to this layer. For instance, if the actuator is connected to the controller through an analog interface, the DCIF driver calls the analog device driver API to connect and actuate using the values received. Here, any signal conditioning can be applied to make the sensor values reliable. Once the data is transmitted, we wait for the actuation process to complete, after which the internal transition function (δint) is activated, and the model passivates until the next request (δext).

The remaining models are defined in a similar way and a description is found in (Jiresal, 2022).

### 3.2 The Informational Layer

The Informational Layer is responsible for making the data more reliable and the system fail-safe. Reliability here refers to the consistency of the data, which is important during data collection and manipulation. Hardware-specific signal conditioning can be applied to the sensor and the Actuator models in the Cyber-Physical Layer, but it would not deal with the information the signals represent but with conditioning the errors/jitters in such signals. We need extra reliability to deal with the informational data obtained from the sensors or being passed to the actuator: data sorting according to the sensor and its type, sensor fusion to fix irregular data, data interpretation, and data assignment to make it presentable/understandable.

The Informational Layer (Figure 2) is defined as a DEVS coupled model with four components: Sorter, Sensor Fusion, Assigner, and Output Commander. These atomic models deal with data manipulation and data fusion. They models are hardware-independent (they do not deal with any of the low-level details managed by the Cyber-Physical Layer) and they can be described as follows:

***- Sorter:*** it is responsible for sorting received data according to sensor types (if multiple sensors are connected in the system) and providing them to the Sensor Fusion Model.

***- Sensor Fusion:*** it helps with the system stability by adding sensor fusion algorithms using data from various sensors to output more robust information to make decisions by the control system.

***- Assigner:*** it organizes the data received and it saves it in a data storage. When receiving a message with fused values, it encapsulates the information in a specific format and it adds more information (units for the sensor data, user-defined details, sensor fusion results, number of faulty sensors, a sensor fusion parameter). The encapsulated message is then sent to a data storage, which is accessible by the Service Layer whenever needs to provide the required informational data to the control system as requested.

***- Output Commander:*** after the control system receives the data, it takes the decision on the commands for the actuators. It receives a sequence container (that may contain multiple actuator commands) from the control system, breaks the container according to the actuator type, and transmits it.

The inputs to the Informational Layer model are given as `TIMESTAMP <SensorName>_Sens<Number> value` where TIMESTAMP is a time value at the input, <SensorName> is the name of a sensor (e.g., Temperature, CO2, etc.), <Number> is the sensor number that can be given to the sensor, and *value* is the sensor reading at that time. There can be multiple instances of the Sensor model for multiple sensors.

The output generated is `{[OUT_TIMESTAMP <SensorName> FusedValue Units UserData], …}` where OUT_TIMESTAMP is a time value, <SensorName> is the name of a sensor, *FusedValue* is a sensor reading obtained after performing sensor fusion, *Units* is a standardized unit for sensor readings and *UserData* is any additional information specified by the user to be included in the message (e.g., faulty sensors).

The inputs to the Service Layer Communicator model are given as `{[TIMESTAMP <ActuatorName> Value Units], …}` where TIMESTAMP is the time value, <ActuatorName> is the name of an actuator (e.g., Motor), *Value* is a standardized value and *Units* are standardized units for the actuator (e.g., RPM).

The output from the Information Layer coupled model is as follows: `OUT_TIMESTAMP ActuatorName ActuatorData` where OUT_TIMESTAMP is a time value, *ActuatorName* is the name of the actuator *ActuatorData* is the lower-level mapped/converted data for the specific microcontroller/microprocessor.

### 3.3 The Service Layer

The Service Layer deals with the communication between the hardware and the control system. It is responsible for providing the following services that connect the hardware to the control system:

- An Application Programming Interface to connect the sensor/actuator to the control system through the Informational and Cyber-Physical Layers.

- Ensure that the hardware and the control system are connected and are transferring data.

- Provide the data required by the control system on request by fetching it from the data storage.

- Redirect the commands from the control system to the hardware using the Informational and Cyber-Physical Layers.

The tasks performed by the Service Layer follow a request-response communication protocol. The Service Layer contacts the control system and checks on what data does the control system need. After it receives a response, it fetches the data from the data storage or sends the current data to the control system according to the request.

The Service Layer is made up of one DEVS model, called the Communicator model, which is responsible for all the communication, synchronization, and data transfer between the Service Layer and the control system. To carry out the communication and the data transfer, the Communicator model has separate communication I/O lines and data I/O lines as shown in Figure 2. The communication of the Service Layer with the control system is hardware-independent and hence, the same formalization can be used with different communication types. The output of the model depends on the state change. Therefore, if the output function contains calls to the functions that enable a different communication technique, it should work for this model. For instance, consider a system with a microcontroller that has an on-chip Wi-Fi transceiver. The control system in this cyber-physical system can be embedded in the same microcontroller or in a different microcontroller that has an on-chip Wi-Fi transceiver as well. In the latter case, the communication with the control system microcontroller will occur through the Wi-Fi connection with or without the internet but using the data packets. For example, it can use an HTTP request-response where the control system microcontroller will host the webserver. That is, the DCIF will communicate using the Service Layer with the control system microcontroller by calling the HTTP methods in the output function to send and receive the data. Due to this flexibility, the DCIF can be easily implemented on any hardware regardless of the communication type or the hardware used

## 4 CASE STUDIES: APPLYING DCIF TO BUILD CPS APPLICATIONS

In the previous section, we discussed the structure of the proposed framework. We implemented the framework in Cadmium (Belloli et al. 2019) and RT-Cadmium simulation tools on real-time embedded hardware. This section discusses the implementation of the framework in Cadmium and RT-Cadmium simulation tools to validate the proposed framework by simulating and implementing the model on a real-time hardware platform. Later in the section, we will also portray the application point of view of the proposed framework along with a case study implementation to support it.

RT-Cadmium is a Real-Time DEVS kernel that uses the real-time clock. It allows users to switch between the simulation and runtime execution in hardware with ease. We used a NUCLEO-H743ZI development board for the deployment of the DCIF. This is an STM32H7 ARM Cortex-M7-based microcontroller. We used temperature and humidity sensors, CO2 sensors, and light intensity/ambient light sensors to build and test the model in real time.

The method we used in the implementation of the Sensor Fusion Model in DCIF is a fusion algorithm that determines the integrated support degree score of each sensor based on principal component analysis (Hongyan 2009). Note that, the sensor fusion algorithm we used can be replaced by any other fusion algorithm in the Sensor Fusion Model or if there is no sensor fusion required in the application, it can be omitted from being used in DCIF.

For both simulation and real-time implementation, we tested the DCIF model with multiple test scenarios, including:

- One sensor of one type. For instance, a temperature sensor.
- Multiple sensors of one type. For instance, 8 temperature sensors.
- Multiple sensors of multiple types. For instance, 4 temperature and 4 CO2 sensors.
  Figure 3 shows the simulation logs for multiple sensors of multiple types connected in the system.

```
00:00:00:000
[CPL_Sensor_defs::out: {Hum_Sens1 75.3}] generated by model CPL_Sensor1
[CPL_Sensor_defs::out: {Hum_Sens4 78.4}] generated by model CPL_Sensor2
[CPL_Sensor_defs::out: {Hum_Sens3 70.9}] generated by model CPL_Sensor3
[CPL_Sensor_defs::out: {Hum_Sens2 79.6}] generated by model CPL_Sensor4
...
00:00:02:000
[] generated by model CPL_Sensor1
…
[] generated by model CPL_Sensor8
[IL_Sorter_defs::out: {1 Humidity 75.3, 2 Humidity 79.6, 3 Humidity 70.9, 4 Humidity 78.4, 1 Tem-
perature 23.2, 2 Temperature 24, 3 Temperature 23.5, 4 Temperature 23.8 }] generated by model
IL_Sorter1
…
00:00:03:000
[] generated by model CPL_Sensor1
…
[] generated by model CPL_Sensor8
[] generated by model IL_Sorter1
[IL_Fusion_defs::out: {Humidity 74.8037, Temperature 23.7695 }] generated by model IL_Fusion1
…
00:00:04:000
[] generated by model CPL_Sensor1
…
[] generated by model CPL_Sensor8
[] generated by model IL_Sorter1
[] generated by model IL_Fusion1
[IL_Assigner_defs::out: {Humidity 74.8037 RH, Temperature 23.7695 °C/°F }] generated by model
IL_Assigner1
...
```

Figure 3: Simulation output for multiple sensors of multiple types in the system.

Figure 3 shows, we consider 4 humidity sensors and 4 temperature sensors for the simulation. We can see that at time 00:00:00:000 the inputs arrive randomly at the DCIF, and the Sorter model sorts the inputs according to the name and number as seen at the time stamp 00:00:02:000. At 00:00:03:000 the fusion model receives the sorted sensor values and sensor fusion is performed for both types of sensor values independently (temperature and humidity). The Assigner at 00:00:04:000 checks for the type of sensors according to the name, adds units to the message, and stores the data in the data storage. Further, we implement the model on real-time hardware NUCLEO board and use DHT11 sensors for temperature and humidity values.

The data stored in the data storage by the hardware is shown in Figure 4.

```
2021-08-11T23:16:08Z,Temperature,27.25,°C
2021-08-11T23:16:08Z,Humidity,65.5,RH
2021-08-11T23:21:12Z,Temperature,27.3,°C
2021-08-11T23:21:12Z,Humidity,65,RH
2021-08-11T23:26:14Z,Temperature,27.5,°C
...
2021-08-11T23:46:26Z,Temperature,27.33,°C
2021-08-11T23:46:26Z,Humidity,65,RH
```

Figure 4: Hardware generated data for multiple sensors of different types.

We conducted a more exhaustive experiment to determine and maximize the reliability of the data by improving the sensor fusion algorithm. The period of the experiment was 24 hours. There were no external sources to affect the temperature and humidity readings drastically and the locations of the sensors were fixed. Faulty scenarios were created manually, and the results were analyzed.

Figure 5 shows a graph of temperature values. We can see that the system remains stable, but there are occasional outliers. For instance, in Figure 5, at the spike locations, we can see that one of the sensors was

disconnected (hence the 0 value for that sensor) but since our system should remain stable as the sensor fusion algorithm we use works even if we disconnect 1/3 of the sensors. These values represented a hard fault situation. The hard faults are the faults in the system that are responsible for the system to fail due to issues related to the used sensor fusion algorithm.

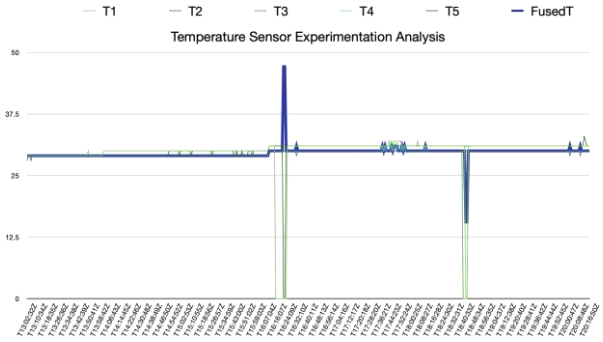


Figure 5: Graph representing the temperature sensor experimentation results.

After the real-time implementation and the simulation of the DCIF model, we carried out a case study that applies the DCIF model to a real-life application. Carleton University comprises more than fifty buildings linked by five kilometers of underground tunnels and a network of roads and pathways. The Carleton Digital Campus is a BIM of Carleton University that permits visualization of the complex data sets. There are several types of data that can be used to evaluate the operation and performance of the buildings including simulated data, experimental data, historical data, or real-time data. The DCIF provides all these types via simulating or through experimentation. Simulated data is used to study various scenarios and to simulate the experimental conditions while the experimental data is directly obtained from measurements performed on the physical model of the building in which real-world scenarios are replicated.

We conducted simulation studies with different temperature readings within the measuring range using DCIF. Faults were deliberately injected in both simulation and experimentation using different scenarios The BIM of a lab in the VSIM building was used to execute the experiments and, as well, a physical model of the lab was fabricated at a scale of 1:20 using laser cutting on acrylic sheets and adding control hardware as shown in Figure 6. The sensors were mounted to the physical model at specific locations on the top to mimic the actual building room.

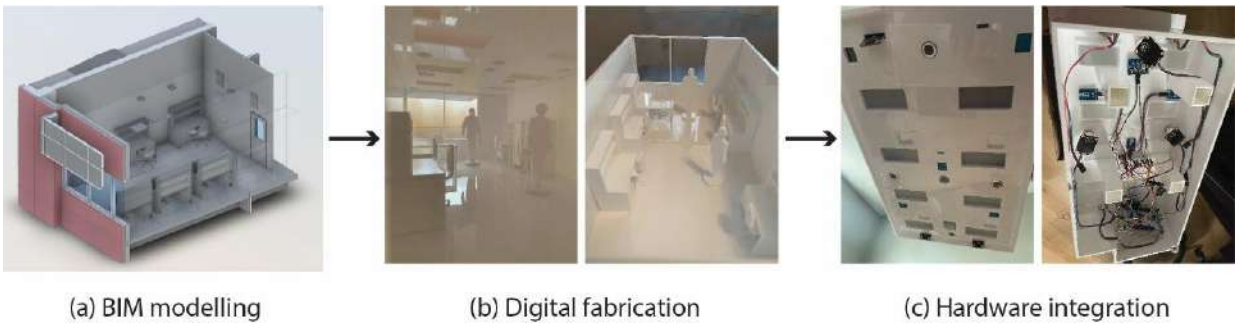


Figure 6: Physical model for experimentation.

Experiments included the collection of data from the sensors using the DCIF and storing it in the data storage. The data collected by using DCIF was used to display it onto a BIM using cloud services. The data storage was provided as an input to the Autodesk Forge software. The sensor readings were perceived by the sensors at a polling rate of 5 minutes (every 5 minutes, a new set of data was fetched from the sensors

and stored in the data storage). The sensor output from the DCIF was then compared with the manually calculated average room temperature from the sensor readings and the comparison was studied.

Figure 7 shows the integration of the data from the sensors along with the fused value. From the experimentation, the ideal room temperature at the given point in time was 27°C. As shown in the figure, the values T1-T5 denote the temperature values at a given time. Faults were manually injected into the system, and hence we can see that the temperatures T1 and T5 are providing 1.92 and 0.00°C, i.e., an erroneous value since they were tampered in the experimentation for some time to provide real-time errors injection. Regardless, the DCIF provided a consistent value of the fused data that is 27°C since the model uses a sensor fusion algorithm to provide a layer of robustness in case of failures. The average temperature has variations due to faulty readings, but the fused value obtained from the DCIF provides a steady temperature value by finding out the integrated support degree score for all the sensors implemented.

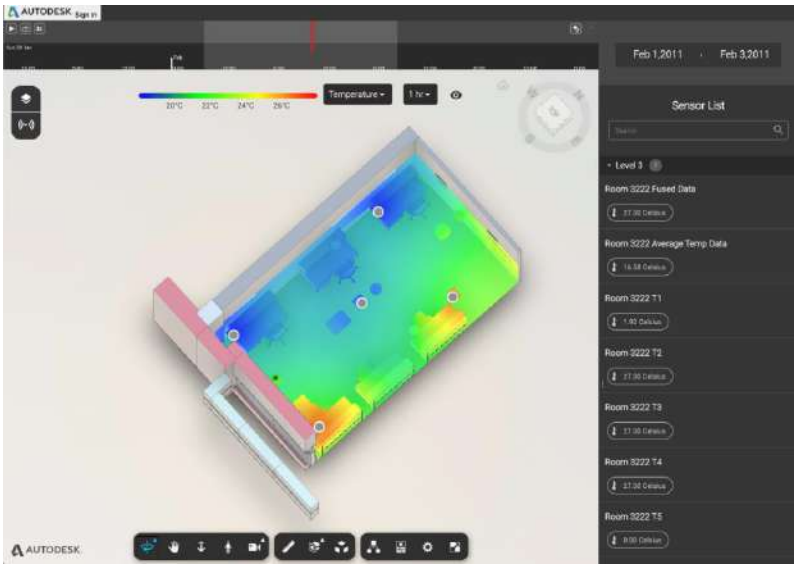


Figure 7: Integration of sensor data with BIM using Autodesk Forge.

The DEVS CPS Interface Framework helped the BIM software obtain the data. We used DCIF to poll the data from the sensors, store it in the data storage in a presentable format and provide it to the BIM for visualization. We could see that the visualization of the data was very intuitive and easy to understand which could help the researchers in the field of building modeling.

## 5 CONCLUSION

We formally defined an architecture for CPS interfaces using DEVS M&S. To support the cyber communication in the model, we used message-based transactions. The model was designed to be extensible, flexible, and easily adaptable for simulation and real-time implementation on target embedded hardware. Additionally, we verified the proposed architecture of the DCIF by implementing a practical application that adapts this architecture. We also illustrated a prototype application using a BIM model of Carleton University campus which has been built for virtual experience and building performance analysis where we use DCIF for data acquisition and storage.

The proposed framework is time-synchronous. This framework perfectly fits the need for non-interrupt-based sensors by using the polling method. But managing asynchronous inputs to this model is not defined and hence any interrupt-based inputs are not taken into consideration. However, in the future, interrupt-based inputs can be added to this framework by adding an additional asynchronous event handle.

In addition, although we implement a Sensor Fusion model in the framework to improve the reliability, hard fault situations can make the system unstable. Some competitive algorithms form conflict situations due to ambiguous sensor inputs. To mitigate this issue, in the future, an implementation of an additional layer of sturdiness that supports the sensor fusion can be done.

## REFERENCES


Astorga, A. M., D. Moreno-Salinas, D. C. Garcia, and J. A. Almansa. 2011. "Simulation benchmark for autonomous marine vehicles in LabView." *OCEANS 2011 IEEE - Spain, Santander, Spain*, pp. 1-6.

Barreras, J. V., C. Fleischer, A. E. Christensen, M. Swierczynski, E. Schaltz, S. J. Andreasen, and D.U. Sauer. 2016. "An Advanced HIL Simulation Battery Model for Battery Management System Testing." *IEEE Tr. on Industry Apps.* 52(6), 5086-5099.

Bergero, F., and E. Kofman. 2010. "PowerDEVS: a tool for hybrid system modeling and real-time simulation." *Simulation: Transactions of the SCS.* 87 (12), pp. 113-132.

Bin L., Xin Wu, H. Figueroa, and A. Monti. 2007. "A Low-Cost Real-Time Hardware-in-the-Loop Testing Approach of Power Electronics Controls." *IEEE trans on Ind. electronics,* 54(2), 919-931.

Boi-Ukeme, J., C. Ruiz-Martin, and G. Wainer. 2020. "Real-Time Fault Detection and Diagnosis of CPS Faults in DEVS." In *IEEE 6th DependSys, Nadi, Fiji, 2020,* pp. 57–64.

Carrijo, D.S., A.P. Oliva, and W.de Castro Leite Filho. 2002. "Hardware-In-Loop Simulation Development." *International Journal of Modelling and Simulation* 22 (3), pp. 167-175.

Castaño, F., S. Strzelczak, A. Villalonga, R. E. Haber, and J. Kossakowska. 2019. "Sensor reliability in cyber-physical systems using internet-of-things data: A review and case study." *Remote sensing* 11 (19), pp. 2252.

Hall, D.L., and J. Llinas. 1997. "An introduction to multisensor fusion." In *Proc of the IEEE*, 85(1)*,* 6-23.

Earle, B., K. Bjornson, C. Ruiz-Martin, and G. Wainer. 2020. "Development of a Real-Time DEVS Kernel: RT-Cadmium." In *2020 Spring Simulation Conference Fairfax, VA.*

Earle, B., K. Bjornson, J. Boi-Ukeme, and G. Wainer. 2019. "Design and Implementation of A Building Control System in Real-Time Devs." *Spring Simulation Conf., Tucson, AZ, 2019.*

Fromel, B.. 2016. "Interface Design in Cyber-Physical Systems-of-Systems." In *11th System of Systems Engineering Conference (SoSE), Kongsberg, Norway, 2016,* pp. 1-8.

Ghaffarianhoseini, A., J. Tookey, N. Naismith, S. Azhar, O. Efimova, and K. Raahemifar. 2017. "Building Information Modelling (BIM) uptake: Clear benefits, understanding its implementation, risks and challenges." *Renewable & sustainable energy reviews* (75), 1046-1053.

Gunes, V., S. Peter, T. Givargis, F. Vahid. 2014. "A Survey on Concepts, Applications, and Challenges in Cyber-Physical Systems." *KSII Tr on Internet and Information Systems, vol. 8, no. 12,* 4242-4268.

Gill, H. 2008. "A continuing vision: Cyber-Physical Systems." In *Fourth Annual Carnegie Mellon Conference on the Electricity Industry.*

Hong, J. S., H. Song, T. G. Kim, and K. H. Park. 1997. "A Real-Time Discrete Event System Specification Formalism for Seamless Real-time Software Development." *Discrete Event Dynamic Systems: Theory and Applications*, pp. 355-375.

Hongyan, G. 2009. "A Simple Multi-Sensor Data Fusion Algorithm Based on Principal Component Analysis." In *ISECS Intl Colloquium on Comp, Comm , Control, and Management, Sanya, China*.

Irwanto, H.Y, and E Artono. 2018. "Correlation of Hardware in the Loop Simulation (HILS) and real control vehicle flight test for reducing flight failures." *J of Physics: Conf Series* 1130(1), 12-14.

Jiresal, R.. 2022. “Formalizing Cyber-Physical System Interfaces Using DEVS”. M.A.Sc. Thesis, Carleton University.

Köhler, C.. 2011. “Enhancing Embedded Systems Simulation A Chip-Hardware-in-the-Loop Simulation Framework”. *Vieweg+Teubner.*

Katipamula, S., and M. R. Brambley. 2005. "Methods for fault detection, diagnostics, and prognostics for building systems - A review, Part I." *HVAC&R research* 11 (1), pp. 3-25.

Kaur, R. K., B. Pandey, and L. K. Singh. 2018. "Dependability Analysis of Safety Critical Systems: Issues and Challenges." *Annals of nuclear energy (120),* pp. 127-154.

Khaitan, S. K., and J. D McCalley. 2015. "Design Techniques and Applications of Cyberphysical Systems: A Survey." *IEEE systems journal* 9 (2), pp. 350-365.

Khaleghi, B., A. Khamis, F. O. Karray, and S. N. Razavi. 2013. "Multisensor data fusion: A review of the state-of-the-art." *Information fusion* 14 (1), pp. 28-44.

Kim, K., and P. R. Kumar. 2012. "Cyber-Physical Systems: A Perspective at the Centennial." In *Proceedings of the IEEE, vol. 100, no. Special Centennial Issue*, pp. 1287-1308.

Lee, D., G. Cha, and S. Park. 2016. "A study on data visualization of embedded sensors for building energy monitoring using BIM." *Intl J of Precision Engineering and Manufacturing* 17 (6), 807-814.

Marwedel, P. 2006. “Embedded System Design”. *New York, Springer US.*

Motawa, I., and A. Almarshad. 2013. "A knowledge-based BIM system for building maintenance." *Automation in Construction* (29), pp. 173-182.

Pratt, A., M. Ruth, D. Krishnamurthy, B. Sparn, M. Lunacek, W. Jones, S. Mittal, H. Wu, and J. Marks. 2017. "H-I-L simulation of a distribution system with air conditioners under model predictive control." *IEEE Power & Energy Soc General Meeting,* Chicago, IL.

Rajkumar, R., I. Lee, L. Sha, and J. Stankovic. 2010. "Cyber-physical systems: the next computing revolution." In *Design Automation Conference, Anaheim, CA, USA*, pp. 731-736.

Ruiz-Martin, C., A. Al-Habashna, G. Wainer, and L. Belloli. 2019. "Control of a Quadcopter Application with DEVS." *Spring Simulation Conference,* Tucson, AZ, USA, pp. 1-12.

Sanislav, T., S. Zeadally, G. D. Mois, and H. Fouchal. 2019. "Reliability, failure detection and prevention in cyber-physical systems (CPSs) with agents." *Concurrency and Computation: Practice and Experience.* 31:e4481.

Volk, R., J. Stengel, and F. Schultmann. 2014. "Building Information Modeling (BIM) for existing buildings — Literature review and future needs." *Automation in construction* (38), 109-127.

Wainer, G.. 2015. "DEVS modelling and simulation for development of embedded systems." In *2015 Winter Simulation Conference (WSC), Huntington Beach, CA, USA,* pp. 73–87.

Zeigler, B. P. 1984. “Multifacetted Modelling and Discrete Event Simulation*.” Academic Press.*

## AUTHOR BIOGRAPHIES

**RISHABH SUDHIR JIRESAL** is currently working as a Developer, Radio Software at Ericsson Canada Inc. He received his M.A.Sc. degree in Electrical and Computer Engineering from Carleton University, Ottawa, ON, Canada. His email address is rishabhjiresal@cmail.carleton.ca.

**GABRIEL WAINER** is a Professor in the Department of Systems and Computer Engineering, at Carleton University (Ottawa, ON, Canada). He is a fellow of SCS. His e-mail is gwainer@sce.carleton.ca.