A FRAMEWORK FOR THE EXTENSION OF DEVS WITH SENSOR FUSION CAPABILITIES

Joseph Boi-Ukeme
Gabriel Wainer

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, Canada
joseph.boiukeme@carleton.ca, gwainer@sce.carleton.ca

ABSTRACT

Sensor fusion is a technique used to combine sensor data and improve the description of the physical property measured by these sensors. Sensors working alone could provide data that is erroneous, incomplete and uncertain. Several algorithms have been developed for sensor fusion, some of which are complex to understand and difficult to implement. The extensive list of algorithms often leaves designers confused about what choices to make as regards which algorithm is best and this discourages them from implementing sensor fusion. In this paper, we propose a generic sensor fusion framework to facilitate the implementation of sensor fusion algorithms using the Discrete-Event Modeling of Embedded Systems (DEMES) which is a model-based development methodology for cyber-physical systems based on discrete-event systems specifications (DEVS).

Keywords: CPS, DEMES, DEVS, Sensor Fusion

1 INTRODUCTION

Sensor fusion is the combining of data from multiple sensors to provide more reliable and accurate information when compared to the information obtained from a single sensor system. Sensor data can be erroneous, incomplete and uncertain, but with the application of sensor fusion, we can tackle these problems thereby providing robustness and reliability, extended coverage, improved resolution and increased confidence within sensor systems (Liggins II et al. 2017). Sensor fusion systems are now widely used in various fields such as sensor networks, robotics, video and image processing, intelligent systems design and many more (Khaleghi et al. 2013).

With the flattering benefits of applying sensor fusion, one would expect that every sensor system employs the use of sensor fusion. However, this is not the case for several reasons. Firstly, sensor fusion has certain drawbacks which include: the fact that you cannot obtain better data than what the individual sensors offer, hence there is no substitute for a good sensor and fusion techniques cannot correct errors in measurement (Akyildiz, et al. 2002); Secondly, there are many fusion algorithms that are rarely used during the design of cyber-physical systems (CPS) because it is difficult to make a choice on what fusion algorithm to use, what advantages they offer and the implementation of these algorithms is not trivial (Lee et al. 2010); Finally, it is difficult to implement sensor fusion and reap the benefits because most design techniques do not provide a framework to implement them thereby discouraging the designers. Incorporating sensor fusion in a design technique is one way to encourage the application of sensor fusion within every sensor system.

SpringSim'20, May 19-May 21, 2020, Fairfax, VA, USA;

Several techniques have been applied in the design of cyber-physical systems such as traditional methods which are heavily dependent on hardware and software design, co-design with hardware in the loop, the use of programming tools, and modeling and simulation. (Khaitan, and McCalley 2015). Among these techniques, modeling and simulation is gaining popularity because of its low development cost, short development time as well as its ability to provide a formal framework for reusability and testing (Niyonkuru and Wainer 2016). If sensor fusion is to be inherent in any design technique, modeling and simulation would be a good place to start. Hence it is important to include sensor fusion in a modeling framework and DEMES is a formal modeling technique based on DEVS with good prospects for extending its capabilities to include sensor fusion.

This paper aims to incorporate a novel generic framework for implementing sensor fusion in DEVS so that, the benefits of sensor fusion can be enjoyed within the modeling phase of the design process. The sensor fusion framework is expanded to account for important aspects of sensor fusion while reducing the level of complexity.

2 BACKGROUND AND RELATED WORK

The design of cyber-physical systems involves three major disciplines: control, systems, and software engineering each of which has its design principles. Each of these disciplines has well established formal methods for design and implementation and it is difficult to adopt any of these disciplines as a universal approach to the design of cyber-physical systems. This has led to an increase in complexity in the design of cyber-physical systems with no unified design method adopted. (Sanislav and Miclea 2012). Although there is no unified method for designing cyber-physical systems, modeling and simulation can be seen as a method that cuts across these three disciplines and can connect the different areas of cyber-physical systems (Rajkumar et al. 2010).

Several modeling approaches and tools have been developed to address the heterogeneity as well as tackle specific applications within the cyber-physical system design domain (Khaitan, and McCalley 2015) which shows that modeling and simulation is gaining popularity among researchers (Lee 2015). Some of the modeling approaches used include metamodeling and metaprogramming methods, formal methods, continuous-time models, finite state machines, and discrete event methods (Khaitan, and McCalley 2015). There have also been several model-based applications for cyber-physical systems, for example, (Douglas et al. 2019) applied modeling and simulation for modeling cyber-physical attacks on water distribution systems using MATLAB while (Jo et al. 2012) developed a model-based fusion method where they combined data from GPS with data obtained from in-vehicle sensors to adequately determine the position of a vehicle in real-time.

Some researchers have made significant attempts to develop formal modeling and simulation techniques for designing cyber-physical systems. For example, (Yue et al 2010) proposed an adaptive discrete event method that incorporates discrete event calculus with the concept of abnormal reasoning to handle unexpected events. Lee (2010) introduced two approaches that complement each other, one to cyberize the physical by introducing software abstractions and the other to physicalize the cyber which introduces physical abstractions around software and communication components. Another method is Discrete-Event Modeling of Embedded Systems (DEMES) (Niyonkuru and Wainer 2016), which is based on DEVS and it is the one adopted in this research because of its advantages such as reliability, model reusability, process flexibility and improved testing (Wainer and Castro 2011). (DEMES is further discussed in section 2.4)

Although there have been significant advances in modeling and simulation as a method for designing CPS', not much can be said about the cyber-physical design methods that incorporate sensor fusion. There has been some work done on MATLAB (Raol 2009) with the introduction of a sensor fusion toolbox to implement a few fusion algorithms. This is very useful but does not take away the complexity and difficulty in deciding what algorithms to use and what happens when new types of sensors are introduced. There has also been some work done on the use of modeling and simulation for specific applications in the sensor

fusion domain which is relatively successful (Khaitan, and McCalley 2015). However, there is still room for improvement. Integrating sensor fusion within a modeling framework would go a long way in reducing complexity and encouraging more CPS' that employ sensor fusion.

In the sections that follow, we describe a few concepts that would make it easier to explain the proposed fusion framework and the choices adopted in terms of configuration and level at which sensor fusion would be done. Some of these concepts include sensor fusion, failure checking, and DEMES.

2.1 Sensor Faults

Failure is the inability of a system/component to perform a required function according to specification while a fault is a condition that could lead to failure (Chen and Patton 2012), therefore it is important to tackle faults early so that they do not lead to failure. In this section, we discuss various common sensor fault types that occur in cyber-physical systems, the list discussed here is not exhaustive but captures the relevant faulty sensor behavior for this work.

Sensor faults occur due to several reasons including aging, battery exhaustion, physical damage and noise (Raposo et al. 2017). Individual sensor faults can affect the performance within a wireless sensor network, studies have reported multiple instances of transient faults in sensor readings within a sensor Network (Sharma et al. 2010).

We define sensor faults as faults that prevent us from capturing the changes in the physical process correctly. The sensor faults discussed here are classified in terms of how the sensors are configured to work; it could be a sensor working on its own or a group of sensors connected to form a sensor network. If we have a sensor working on its own, we call it an individual sensor and if we have a group of sensors working together, we call it a sensor network. It is important to note that there could be faults in any of the two cases described above and some faults are common to both cases. We classify these faults as follows:

2.1.1 Individual sensor faults

For an individual sensor, typical faults could be classified based on the observed pattern of the sensor behavior or based on the reason why the sensor is faulty. In terms of the pattern of the fault, we classify them as follows

- **Bias:** The physical characteristics of these sensors change over time, resulting in different behavior when compared to expected behavior, this is known as bias. In some cases, the bias is an offset from the original sensor, meaning it still obeys the normal sensor behavior, but the values are either above or below the expected value, while in other cases, the bias does not follow the original behavior. Internal sensor bias could increase based on usage time. Sensor bias is a sensor fault and we could have fixed bias or constant bias (Sharma et al. 2010).
- **Out of Range:** Sensors exhibit a proportional response during operation within set parameters and this is called the normal range of the sensor, outside this set parameter, they are now out of range. Out of range faults occur when a sensor behavior deviates from what it should be when compared to the normal range. It could occur as a result of a problem within the sensor itself or as a result of extreme environmental factors (Sharma et al. 2010, Ganeriwal et al. 2008).
- **Distortion as a result of noise:** Sensors typically respond to changes in physical process and output an electrical signal, the electrical signal outputted can be distorted by noise. This noise signal is any unwanted signal that degrades the integrity of the sensor signal. Noise can lead to sensor faults and it is important to deal with these kinds of noise (Sharma et al. 2010, Tönshoff and Inasaki 2001).

As we can see from the foregoing, sensor faults can be observed from the sensor readings by comparing them with an expected value, however, that does not show the underlying cause of the sensor fault. Some common reasons why sensor faults occur include (Sharma et al. 2010).

- Aging, battery exhaustion, physical damage, and noise

- Calibration Faults
- Connection Faults

2.1.2 Sensor Networks

For sensor networks, in addition to the faults that occur with individual sensors, typical faults that could be seen from the systems point of view called system-view faults or by observing the data known as data-centric faults. The typical faults within these categories include:

Data-Centric Faults: A fault determined based upon observing data from a sensor network. Typical faults could be any of the following: Outliers, Spikes, Stuck-at, or Noise (Ni et al. 2009).

System-View Faults: Faults are classified based on the reason why it occurred. They include Calibration Faults, Connection/Hardware Fault, Low battery and Environment out of range (Ni et al. 2009).

2.2 Sensor Fusion Configurations and Algorithms

To combine sensors to achieve sensor fusion benefits, we need to decide how the sensors would be connected, i.e., we need to decide the sensor fusion configuration. There are different sensor fusion configurations (Liggins II et al. 2017):

- **Complementary fusion:** Sensors work independently and complement each other giving a more complete view of the object or parameter being observed.
- **Competitive configuration:** Sensors get an independent measurement of the same property, thereby building confidence in the measured data. The competitive configuration is often used in fault-tolerant systems.
- **Cooperative configuration:** Sensor fusion could combine information from independent sensors that ordinarily would not be available from an individual sensor in a cooperative configuration. For example, combined different two-dimensional images from two-dimensional cameras to produce a three-dimensional image.

There are several ways to classify sensor fusion methods or algorithms, but for this work, we stick with classifying them according to the level in which sensor data is fused. This is called the three-level categorization. (Elmenreich 2002):

- **Low-Level Fusion:** It is also known as raw data fusion and it fuses several sources of raw data to create new data that is expected to be more informative than the input data. It is generally simple to implement and can be done in real-time but is heavily sensor dependent and sensitive to noise and interference.
- **Intermediate-Level Fusion**: It is also known as feature level fusion and it fuses object features like edges, lines, textures that can be used for detection. It is generally less dependent on the sensor but complex in dealing with these features.
- **High-Level Fusion**: It is also called decision level fusion and it combines decisions from several experts. Generally, some processing is done at the sensor level before data is fused. At this level, we can deal with any type of information including rules, since some level of decision is made at the sensor level.

2.3 DEVS formalism

DEVS formalism is a formal modeling methodology that is based on systems theory. In DEVS, a system can be modeled by specifying atomic and coupled models (Zeigler, Kim, and Prähofer 2000). An external input received by such a system would cause a state change after a time delay elapses. The DEVS decomposes complex systems into atomic and coupled models, where the atomic models specify the

behavior and the coupled model specifies the structure (Wainer 2009). DEVS provides a rich structural representation of components and can explicitly specify the timing, which makes it adaptable for real-time systems. Various real-time systems have been successfully developed using DEVS (Song and Kim 2005; Earle et al. 2019; Ruiz-Martin et al. 2019). DEVS is developed in such a way that other discrete event formalisms can easily be mapped into DEVS (Mittal and Mart 2013, Wainer 2019). In addition to the well-developed mathematical foundation of the DEVS formalism, it also proposes an abstract simulation algorithm independent of the models developed. DEMES uses discrete event methodology and DEVS is a well-known example.

2.4 Discrete-Event Modeling of Embedded Systems (DEMES)

DEMES analyzes the cyber-physical system and allows the original model to be part of the final product while using formal models to study the interaction with the physical environment. This is done by gradually replacing models with hardware surrogates and new software components without changing the original models. Migration can be done in stages. After thorough testing, models are incorporated into the target environment and reused throughout the development process (Niyonkuru and Wainer 2016). In DEMES, a cyber-physical system of interest would be modeled using the DEVS theory by properly describing the systems requirements and the environment in which it operates after which these models can be formally validated. Simulations and tests can be run by using sensor data and when the behavior of the model(s) is satisfactory, the models can be implemented on a target platform with real sensors (Niyonkuru and Wainer 2016).

DEMES has a structured development cycle. The steps involved in the DEMES development cycle are described in Figure 1.

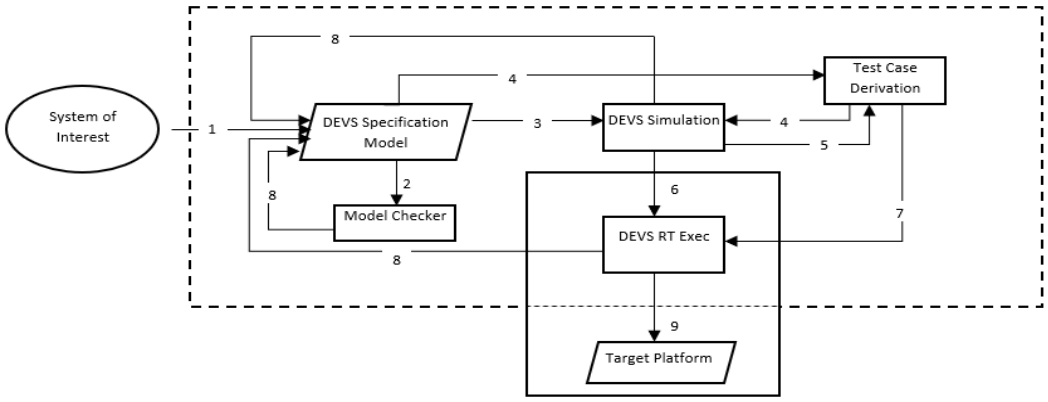

Figure 1: DEMES Development cycle (Wainer and Castro 2011).

In Figure 1, the first step is to identify the cyber-physical system of interest, after identification of the system, the system is modeled and transformed into DEVS models (atomic and coupled models). The next step is to check these models and run simulations to ensure the models valid. The DEVS models can also be used to derive test cases, which can be used for more simulation studies. The valid models are then implemented on a target platform through the DEVS real-time executive and if there were parts of the models that were unverified within the simulation environments, they can be tested on the target platform. If further modifications are required, we can go back to the same DEVS models and implement these changes (Niyonkuru and Wainer 2016).

3 PROPOSED SENSOR FUSION FRAMEWORK

In this section, we describe the proposed sensor fusion framework and the various requirements and design considerations.

3.1 Design Considerations

The sensor fusion framework will receive inputs from different sensors and give out an output with the measured property after the fusion algorithms are applied.

In section 2, different configurations on how sensors can be fused were discussed and the various algorithms to implement sensor fusion would depend on how the sensors are configured. Therefore, configuration selection is an important consideration for our proposed sensor fusion framework. The sensor fusion algorithm is the core of any sensor fusion system; therefore, we also have to consider algorithm selection within the proposed framework.

It has also been established that the quality of the sensor inputs to a fusion system greatly affects the outcome of the fusion process with the possibility of degrading its performance, therefore our proposed framework would consider the option to deal with corrupt sensor data and guarantee a fused sensor output with better quality than the input.

Another important consideration is how to integrate the sensor fusion framework with DEVS. The first possibility would be to modify the input within the DEVS formalism. This is possible but brings few challenges such as backward compatibility to the original DEVS (which is a well-established method) and an increased level of complexity for the modeler which defeats the purpose of our proposed framework. The second possibility would be incorporating sensor fusion within a Real-Time (RT) DEVS simulator. In this case, the entire fusion process would be hidden from the modeler making it simpler and we would also preserve the DEVS formalism and backward compatibility with other DEVS models. The second case presents more advantages and has been adopted for this framework.

3.2 Requirements

The requirements that describe the functionalities of the sensor fusion framework are as follows:

- **Works for all applications:** The sensor fusion block is designed to be generic and can be used for any application irrespective of the type of sensor employed. The framework would take advantage of sensors being either analog (output being a change in voltage) or digital (output being a sequence of bits).
- **Flexible and extensible:** The framework would provide a collection of different fusion algorithms that cover the range of application of sensor fusion and allow for the user to implement different algorithms if needed
- **Ensure the core requirements of sensor fusion are met:** The fusion block is designed to ensure the core advantages of sensor fusion which include robustness and reliability, extended coverage, improved resolution, and increased confidence are met. Output values should be more reliable than the input property value. It should be easy to incorporate new algorithms to the fusion block and we should have a way of handling missing measurement and incorrect sensor data.
- **Receive a set of inputs and give a single output at an instant of time:** The idea is to ensure that the modeler would not face increased complexity with the addition of the sensor fusion framework, it would be the same level of complexity assuming the modeler was working on a single sensor system.
- **The output should be identical to the output from a single sensor for property in question:** An identical output data type from the sensor fusion block with a single sensor system would reduce complexity.
- **Properties should have unique identifiers and timestamp:** This would help us keep track of the sensor data that has been fused.

3.3 Description of the Sensor Fusion Framework

In this section, the proposed fusion framework developed to meet the requirements outlined in section 3.2 is described. Let us assume we have a system of sensors (S_1…. Si) measuring the same property and these sensors are inputs to our sensor fusion block as shown in Figure 2. In Figure 2, we see that the fusion framework has two layers, the first layer is the sensor layer which has several sensors (S1…Si), and the fusion layer which is made up of four blocks namely: configuration selection, fusion algorithm layer, failure checking layer, and fault-tolerant layer. The configuration layer is the input decision for the fusion algorithm selection. The failure checking layer works with the inputs and outputs of the sensor fusion layer as its input and it serves as the decision input for the fault tolerance layer.

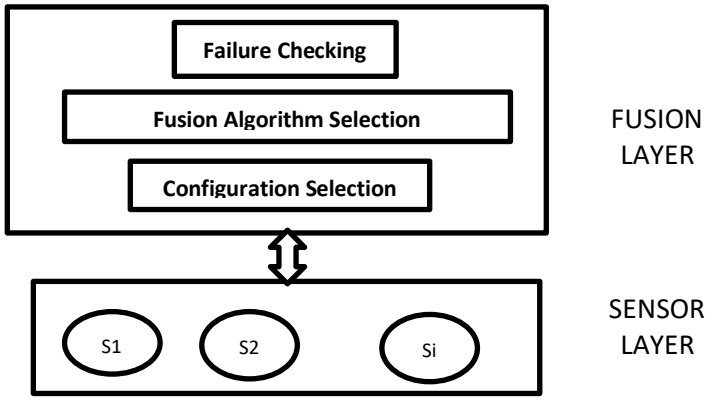

Figure 2: Proposed Sensor Fusion Framework.

3.3.1 Sensor Layer

In the sensor layer, we have a group of sensors measuring the same property, these may be the same sensor type or different sensor types, but the key point is that the property they measure must be the same. For example, we can sense temperature changes using a thermistor or a thermocouple. If we are to fuse different data obtained from sensors measuring different properties, then we would have more than one sensor layer.

3.3.2 Fusion layer

The fusion layer has the following four blocks:

Configuration Selection

As discussed in section 2, we have different configurations for combining sensor data. In this layer, the modeler can select one of three possible configurations namely, Competitive, Cooperative or Complementary. The choice of the configuration would depend on the objective of the sensor fusion.

The configuration layer would be the input decision for the algorithm layer. In this layer, more than one configuration can be chosen at different times. For example, a modeler may want to do a two-phase configuration for sensor fusion; phase 1 can be competitive to ensure reliability and accuracy while phase 2 can be complementary for completeness.

Fusion Algorithm Selection

In this layer, we have a collection of fusion algorithms based on the sensor configuration. For this framework, to be generic, the high-level or decision level of sensor fusion would be adopted. The algorithms would be classified based on configuration, and when the modeler selects a configuration with an objective

in mind then the decision level fusion algorithms would be streamlined to fulfill those objectives. The decision level algorithm has been chosen for several reasons:

1) It allows for the fusion block to be general-purpose.
2) It avoids the need for low-level interpolation.
3) Because the decision would be made at the sensor level the sensor data would be checked twice improving the confidence in the fused sensor data.
4) It would allow for an easy combination of sensor data for control and allow new sensors to be incorporated into an existing design easily.

The algorithm selection layer also allows for more than one algorithm to be implemented. For example, a weighted fusion algorithm can be used to fuse sensor data and a filtering algorithm can be used to obtain true values. The output of the fusion algorithm selection layer would have one sensor value at an instance of time which would be similar to the output that would have been obtained from a single sensor.

Failure Checking

In this layer, we check the inputs that have been fused to determine problems with input data and check the quality of the fused sensor output at different time intervals to be sure that they are representative of the physical property being observed.

A proposed failure checking function would be checking individual sensor values for n consecutive time intervals separated by Δt (the value of Δt is obtained from experiments based on the maximum time in which a sensor change would be expected) and if the value never changed then there is likely a sensor fault.

Other failure checking mechanisms can also be employed for example checking the absolute difference of sensor values and weighted average values within an acceptable limit defined by the modeler. The main idea is that this layer would be able to identify a fault in the sensor layer and/or the fusion layer.

3.4 Sensor Fusion Framework within a CPS control system

A DEVS based CPS control system is shown in Figure 3. The control system is made up of the environment that is to be controlled, the sensors to observe changes in certain physical properties within the environment, the sensor fusion block described earlier, a DEVS-based controller, an actuator used to switch the devices responsible for maintaining a set point for the property to be controlled.

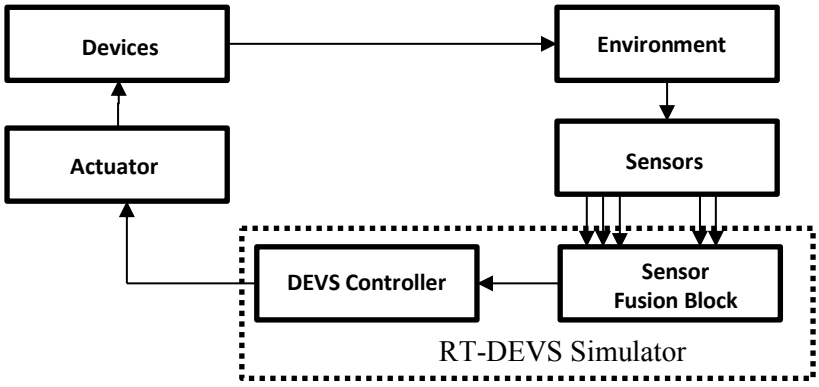

Figure 3: A Control System.

With this proposed framework, we would have a RT-DEVS simulator that can perform sensor fusion and perform tasks to control the entire system. As seen from the control system, with a RT-DEVS simulator, we can implement a DEVS-based controller that can run directly on hardware seamlessly. With sensor fusion integrated, it would be possible to deploy new sensors seamlessly both in the models and on the target platform as old models can be reused. This is a very attractive feature because sensors are evolving every day.

4 IMPLEMENTATION OF THE SENSOR FUSION FRAMEWORK

To illustrate the applicability of the sensor fusion framework, we implemented an example. For this example, the competitive configuration is adopted; the sensor fusion algorithm implemented is derived from the concept of principal component analysis (PCA), and a simple failure checking algorithm is also incorporated; the failure checking algorithm ensures that sensor data should also follow a correct trend, for example, it should have a linear response and not have random spikes except it is out of range.

The algorithm takes raw data from the sensors as inputs within a timeframe, performs principal component calculations and returns a single output at an instant of time. The value generated by the algorithm is representative of the correct raw data from the sensors even though there may be erroneous readings on some sensors. The algorithm is based on the paper presented in (Hongyan 2009); a summary of the steps in the algorithm is as follows

- Compute the distance function
- Use distance function to create a support degree matrix
- Calculate the eigenvalues for each sensor support degree matrix
- Use these eigenvalues to compute the principal components
- Compute the integrated support degree score for each sensor
- Discard invalid sensor reading
- Compute fused output by the weighted average sum of valid sensors

The failure checking block checks for the following conditions:

- No change in sensor output after a specified time elapses (sensor value is stuck)
    - Stuck Time Interval (-s): Specifies how long (in minutes) beyond which, the sensor is stuck, and no longer valid. Depending on the property in question, there is a time for an expected change s (for example if it is temperature, we can expect a change every 1 hour, therefore, s = 1 hour)
- Sensor values lie outside a prescribed range (outlier) and Spikes in sensor data
    - Lower Limit Value (-l): Specifies the lower limit below which the sensor will be marked out of range.
    - Upper Limit Value (-u): Specifies the upper limit above which the sensor will be marked out of range.

The user of the algorithm can modify these parameters to suit the desired application.

Algorithm Example

For this example, we define the following parameters in the simulation; Stuck Time Interval as 10 minutes with sensor values range from 1 to 10 (s = 10 minutes, l = 1, and u =10). The sensor readings considered are shown in the following sensor readings in Table 1.

Table 1: Sensor reading from 8 sensors at 4 different times.

Time	S1	S2	S3	S4	S5	S6	S7	S8
1:20:00 PM	1.2	1.3	1.4	-	15	-	17	-
1:50:00 PM	1.2	1.3	1.4	1.5	1.2	1.3	11	-
2:00:00 PM	1.2	1.3	-	1.5	0	17	1.4	1.5
2:10:00 PM	1.2	1.3	1.5	1.5	1.2	1.3	1.4	1.5

The data from the table shows the different sensor readings and a few other readings that are out of range. After implementing the fusion algorithm and the failure checking block, we tested it using the data obtained from table 1. The sensor data output obtained is described in Table 2 and Figure 4.

Sample Output

The output logs obtained from our fusion block gives us a timestamp, the fused sensor value at that time, and the statistics for each sensor's contribution, a sample of this information is shown in Table 2.

Table 2: Sample Sensor Fusion Output.

Time	Sensor	Value	Status	
1:50	S3	1.4	Valid	Time Stamp =2:00pm
1:50	S4	1.5	Valid	
2:00	S5	1.2	Valid	Fused Sensor Value =1.3857
2:00	S6	1.3	Valid	
2:00	S7	1.4	Valid	
2:00	S8	1.5	Valid	
1:20	S1	1.2	Stuck	
1:20	S2	1.3	Stuck	

Table 2 shows the fusion block output at 2:00 pm. The data involved includes data from sensors reporting their value at time T = 2:00 pm and also data from sensors that have not changed their values from the previous times of reporting. This data allows us to compute the fused sensor value using the algorithm and also to check for failures. As can be seen from the table, S1 and S2 are stuck at a value because they did not change within the expected period defined. The other results from the input sensor data can be explained using the plot in Figure 4.

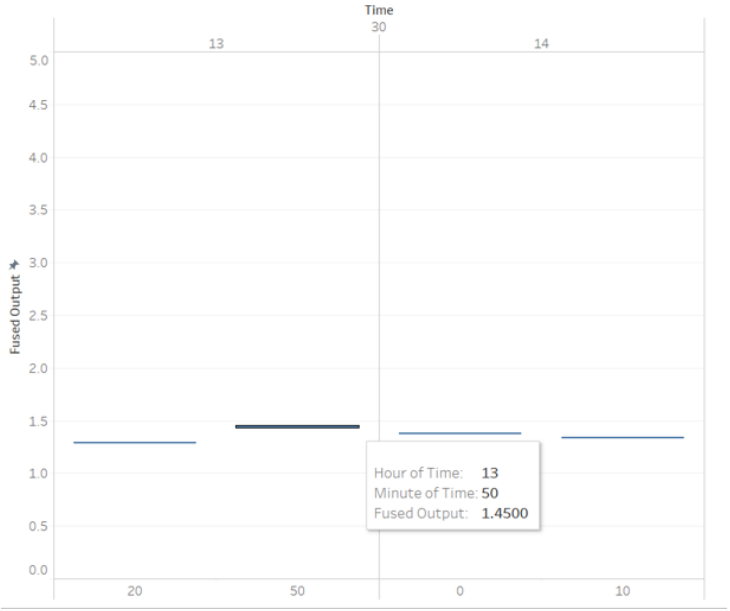

Figure 4: Sensor Fusion Output.

Figure 4 shows the sensor fusion output for different timeframes. The horizontal axis shows time and the vertical axis is the fused sensor value. If we compare the different values of the input sensor with the fused output, we can see that we have a representative sensor output irrespective of the fact that there are some missing measurements and some sensor values out of range.

5 INTEGRATION OPTIONS TO A RT-DEVS SIMULATOR

To incorporate the sensor fusion framework into a RT-DEVS simulator, we develop the sensor fusion algorithms and failure checking algorithms as DEVS models that can easily be coupled with other DEVS models and paves the way for combining more than one algorithm within a sensor fusion block. This ensures that all sensor fusion models would be independent of the simulator and can be backward compatible with existing models. The structure of how the framework fits with a RT-DEVS simulator is illustrated in Figure 5.

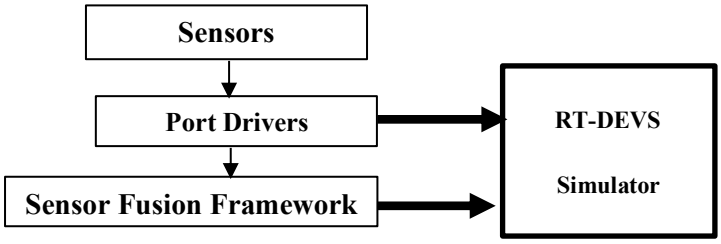

Figure 5: Integration of Sensor fusion framework with a DEVS Simulator.

From Figure 5, we can see where the sensor fusion framework fits with the simulator. The integration is done such that when there is a single sensor system that does not need the fusion framework, the simulator can obtain sensor inputs directly from the sensors through the port drivers.

6 CONCLUSION AND FUTURE WORK

We have highlighted the challenges of sensor fusion for cyber-physical systems and discussed some of the efforts that have been put as regards modeling and simulation. A gap for improving a modeling and simulation technique has been identified and a framework for the inclusion of sensor fusion in DEVS has been proposed. This framework would make the implementation of sensor fusion less complex and easily accessible to the modeler as an option in a RT- DEVS simulator. In our future work, we would explore more algorithms for sensor fusion and failure checking. The framework would also be implemented on target hardware to allow us to perform more experiments to evaluate the performance of this framework.

REFERENCES

Akyildiz, I. F., W. Su, Y. Sankarasubramaniam, and E. Cayirci, 2002. "A survey on sensor networks". *IEEE Communications Magazine*, *40*(8):102-114.

Chen, J., & Patton, R. J. 2012. "Robust model-based fault diagnosis for dynamic systems" *Springer Science & Business Media*. (Vol. 3)

Douglas, H. C., R. Taormina, and S. Galelli. 2019. "Pressure-Driven Modeling of Cyber-Physical Attacks on Water Distribution Systems". *Journal of Water Resources Planning and Management,* 145(3)

Elmenreich, W. 2002. *An Introduction to Sensor Fusion*. Austria: *Vienna University of Technology.*

Ganeriwal, S., Balzano, L.K. and Srivastava, M.B., 2008. Reputation-based framework for high integrity sensor networks. ACM Transactions on Sensor Networks (TOSN), 4(3), p.15.

Hongyan, G. A. O. 2009. A simple multi-sensor data fusion algorithm based on principal component analysis. In *ISECS International Colloquium on Computing, Communication, Control, and Management* (Vol. 2, pp. 423-426). IEEE.

Jo, K., K. Chu, and M. Sunwoo, 2012. "Interacting Multiple Model Filter-Based Sensor Fusion of GPS with In-vehicle Sensors for Real-Time Vehicle Positioning". *IEEE Transactions on Intelligent Transportation Systems*, *13*(1):329-343.

Khaitan, S. K., and J. D. McCalley. 2015. "Design Techniques and Applications of Cyberphysical Systems: A Survey". *IEEE Systems Journal*, 9(2):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):28-44.

Lee, E. 2015. The past, present, and future of cyber-physical systems: A focus on models. Sensors, 15(3), 4837-4869.

Lee, E. A. 2010. "CPS Foundations". In *Design Automation Conference,* June 13th-18th, Anaheim, California, USA, 737-742.

Lee, H., B. Lee, K. Park, and R. Elmasri. 2010. "Fusion Techniques for Reliable Information: A Survey". *International Journal of Digital Content Technology and its Applications*, *4*(2):74-88

Liggins II, M., D. Hall, and J. Llinas, Editors. 2017. "*Handbook of Multisensor Data Fusion: Theory and Practice"*. CRC Press.

Mittal, S. and Mart, J.L. 2013. *Netcentric system of systems engineering with DEVS united process*. CRC Press.

Ni, K., Ramanathan, N., Chehade, M.N.H., Balzano, L., Nair, S., Zahedi, S., Kohler, E., Pottie, G., Hansen, M. and Srivastava, M., 2009. Sensor network data fault types. *ACM Transactions on Sensor Networks (TOSN)*, *5*(3), p.25.

Niyonkuru, D., and G. Wainer, 2016. "A kernel for Embedded Systems Development and Simulation Using the Boost Library". *In Proceedings of the Symposium on Theory of Modelling & Simulation Society for Computer Simulation International*, April 3rd-6th, Pasadena, California, 13.

Rajkumar, R., I. Lee, L. Sha, J. Stankovic. 2010. "Cyber-Physical Systems: The Next Computing Revolution". *In Design Automation Conference,* June 13th-18th, Anaheim, California, USA, 731-736.

Raol, J. R. 2009. *Multi-Sensor Data Fusion with MATLAB®*. 1st ed. Boca Raton: CRC Press.

Raposo, D., Rodrigues, A., Silva, J. S., & Boavida, F. 2017. "A taxonomy of faults for wireless sensor networks". *Journal of Network and Systems Management*, *25*(3), 591-611.

Sanislav, T., and L. Miclea. 2012. "Cyber-Physical Systems-Concept, Challenges and Research Areas". *Journal of Control Engineering and Applied Informatics,* 14(2):28-33.

Sharma, A.B., Golubchik, L., and Govindan, R., 2010. Sensor faults: Detection methods and prevalence in real-world datasets. ACM Transactions on Sensor Networks (TOSN), 6(3), p.23.

Tönshoff, H.K. and Inasaki, I. eds., 2001. Sensors in manufacturing (Vol. 236). Weinheim: Wiley-VCH.

Wainer, G. 2019. *Applying Modelling and Simulation for Development of Embedded Systems.* Proceedings of SummerSim 2019. Berlin, Germany.

Wainer, G., and Castro, R. 2011. "DEMES: a Discrete-Event methodology for Modeling and simulation of Embedded Systems" *Modeling and Simulation Magazine, 2*, 65-73.

Yue, K., L. Wang, S. Ren, X. Mao, and X Li. 2010. "An Adaptive Discrete Event Model for Cyber-Physical System". In *Analytic Virtual Integration of Cyber-Physical Systems Workshop,* November 30th, San Diego, California, USA, 9-15.

Zeigler, B. P., T. G. Kim, and H. Prähofer, 2000. *Theory of Modeling and Simulation: Integrating Discrete-Event and Continuous Complex Dynamic Systems*. 2nd ed. San Diego, California: Academic Press.

AUTHOR BIOGRAPHIES

JOSEPH BOI-UKEME is pursuing a Ph.D. in Electrical and Computer Engineering at Carleton University where he researches on Cyber-physical Systems. His email address is joseph.boiukeme@carleton.ca.

GABRIEL WAINER is a Professor at the Department of Systems and Computer Engineering at Carleton University. He is a Fellow of the Society for Modeling and Simulation International (SCS). His email address is gwainer@sce.carleton.ca.