BROOKS-IYENGAR ALGORITHM IN PUB/SUB ARCHITECTURE USING THE DEVS FORMALISM

Iman Alavi Fazel[a] and Gabriel Wainer[a]

[a]Carleton University, Canada
imanalavifazel@cmail.carleton.ca, *gwainer@sce.carleton.ca*

ABSTRACT

Technologies that interact with the physical world rely on various types of sensors to measure environment variables. However, sensors can become defective due to aging and harsh conditions, leading to inaccurate readings and incorrect decisions. To address faulty sensors, a distributed network with replicated sensors can be used. In this research, we combine modeling and simulation to develop these applications, proposing the Brooks-Iyengar algorithm for sensor replication in a publish/subscribe architecture. Our approach demonstrates how a modeling and simulation platform can develop embedded applications that achieve multisensory fusion and inexact agreement between nodes. Using the Discrete Event System Specification (DEVS), our models are utilized both in simulation environments and on hardware.

Keywords: fault-tolerant algorithms, Brooks-Iyengar algorithm, IoT, DEVS.

1 INTRODUCTION

Sensors are an integral part of technologies that interact with the physical world. Examples of these systems include the Internet of Things (IoT), cyber-physical systems (CPS), and wireless sensor networks (WSNs). Sensors, however, are subject to malfunction due to various factors, including environmental, aging, manufacturing problems, and other variables that can affect them. To prevent the system from deviation when sensors become faulty, multiple sensors can be deployed redundantly. With the presence of redundant sensors and considering that the number of faulty sensors is within a certain limit, the systems can combine or 'fuse' data from multiple data sources to compute values that are free from faulty readings [1]. Furthermore, the computed results are more accurate than the values the system would get from a single sensor's data. The combination, or fusion of sensor data can be achieved through a class of algorithms called 'multisensor fusion'.

In addition to employing a device with redundant sensors, a network of devices, each equipped with its own set of sensors can be deployed in an environment. The fused values can then be used for decision-making on the devices or be transmitted to the network as devices' final values. The network of devices adds a new dimension to the robustness of the system; however, the distributed nature of nodes causes them to retrieve data, fuse, and report different values in irregular intervals. In this context, the nodes can become synchronized between reading intervals by "agreeing" on common values using 'inexact agreement' algorithms [2]. These methods, which are variants of algorithms devised for the Byzantine Generals problem, are useful for dealing with redundancy in distributed systems [3]. Inexact agreement algorithms, in particular, perform multiple rounds of message exchange among the nodes, causing the nodes' values to converge to the same values. Although the result of the message exchange might cause some nodes to reach less accurate values than their initial reading, achieving consensus, considering the errors are in a certain bound, is often a desirable outcome in distributed systems.

A distributed network of nodes each equipped with a set of sensors can use the Brooks-Iyengar, a hybrid algorithm, to perform multisensor fusion on their replicated sensor readings, as well as to achieve inexact

Proc. of the 2024 Annual Simulation Conference (ANNSIM'24), May 20-23, 2024, American University, DC, USA
C. Ruiz-Martin, B. Oakes and R. Cárdenas, eds.
©2024 Society for Modeling & Simulation International (SCS)

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

agreement with other nodes within the network [4]. In this work, we applied this algorithm in a cyber-physical system application in which the nodes are communicating via the pub/sub architecture. Furthermore, the methodology is based on using the Discrete Event System Specification (DEVS) formalism for modeling, simulation, and execution in real time. Adopting DEVS allowed us to develop a set of models and run them in a simulated environment, as well as on the actual devices for their operation at runtime. The execution of the models on the devices is achieved by deploying the model alongside a real-time DEVS engine on an embedded hardware platform. Furthermore, DEVS facilitated the model-driven development (MDD) of this IoT application. The idea is that one can design the intended behavior of IoT devices using high-level semantics (rather than implementation-dependent code), and consequently, any hardware that supports the DEVS simulator will be able to run the models seamlessly. Lastly, a rich set of verification and validation (V&V) techniques has been developed for DEVS model which can be leveraged when using DEVS for MDD approaches [5, 6, 7].

The rest of the paper is organized as follows. In Section 2, we discuss recent research in sensor fusion and agreement problem algorithms for sensor networks, as well as a brief description of DEVS. Section 3 discusses the architecture and the DEVS models that were developed for our methodology. Section 4 provides some information on the implementation of the DEVS models in this study. The results of a case study comprising multiple sensors, each positioned in different locations, are presented in Section 5. Lastly, Section 6 summarizes the conclusions of this research and discusses future directions.

2 RELATED WORK

Multisensor fusion refers to algorithms that take data from various sources and combine them to produce more accurate and reliable data [1]. These algorithms can be classified from different perspectives [8]. One such classification is based on the level of abstraction of the data that the algorithms perform their computations on. Low-level fusion algorithms often perform real-time computations on raw sensor data to produce higher-quality data of the same type. At the same time, high-level techniques involve computing symbolic representations of different data that could have been taken from various locations of the environment. The authors further conducted a classification based on the nature of the input and output of the algorithms. For instance, an algorithm may take data and fuse them to produce the same form of data but with higher accuracy, or instead, they may produce features by combining them together. Lastly, fusion algorithms were categorized based on the relationship of the nodes which can be complementary, redundant, or cooperative. Surveys [9], [10], and [11] summarize some of the commonly used techniques for sensor fusion in the context of IoT and sensor networks. These techniques include methods such as Bayesian networks, Kalman filters, and knowledge-based methods. Examples of these works include [12] where authors used an extended Kalman filter (EKF) and a machine learning approach to estimate the position of the vehicle using GPS and a combination of odometer and gyro-meter. In [13], authors applied several statistical models such as Linear Discriminant Analysis (LDA) and Random Forest (RF) to detect occupancy of a room using light, temperature, humidity, and CO2 data. Marzullo [14] developed a sensor fusion algorithm that receives replicated sensor readings and based on the feasible range assigned to each value, computes a new range in which the true value of the environment lies. In their work, physical sensors which capture data from the environment are called “concrete sensors”. The readings of concrete sensors are then forwarded by “abstract sensors” in which a range is associated with their readings.

A large class of sensor-based systems that are deployed in practice form a distributed network of nodes. In this context, a recurring problem involves achieving consensus among the nodes considering that some of them may report conflicting information with respect to other nodes. Problems of this type have been extensively studied in literature by formulating the problem to what is known as the Byzantine Generals Problem [3]. This problem involves a hypothetical siege of a city in which multiple generals are situated around a city and they must reach a decision to either attack the city or retreat their army. In this scenario, the generals, as well as messengers, could be traitors and can intentionally receive and transmit contradictory messages to other generals. In sensor networks, when a node sends a message, the system is

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

said to tolerate Byzantine fault if (i) all non-faulty or "loyal" nodes agree on the same message and (ii) the sending node was non-faulty, all non-faulty components agree on the same message. Several works applied Byzantine agreement algorithms for IoT and sensor-based networks. These works include [15], where authors developed an agreement protocol for cluster-based wireless sensor networks (CWSN). In CWSN, the network consists of a set of clusters, each with multiple nodes and one cluster head. The cluster head manages the state and connections of the nodes in its cluster. Their protocol, moreover, achieves agreement with the minimum number of message exchanges. In [16], the authors developed an agreement algorithm for an IoT platform that follows fog and cloud architectures. In their protocol, the fog nodes first establish consensus among the sensor layer nodes connected interacting with them, and eventually, agreed values are forwarded to the cloud layer. The complexity and correctness of their protocol are also presented in their work. A variant of the Byzantine Generals Problem, named, 'inexact agreement' was studied in [2]. In this version, the nodes report real values, and the goal for the nodes is to reach values that are in a certain bound of each other. The authors developed two algorithms for this problem: namely, the Fast Convergence Algorithm (FCA) and the Crusaders Convergence Algorithm (CCA). Their algorithms can cause the nodes to exchange rounds of message exchange which causes their value to converge as long as less than one-third of the nodes are faulty. The authors later used the FCA algorithm to propose a protocol that can be used for clock synchronization.

The Brooks-Iyengar algorithm [4] is a hybrid between the multisensor fusion algorithm developed by Marzullo [14], and the FCA algorithm that was devised for an inexact agreement problem [2]. This algorithm was applied in previous works such as [17], where it was used in a transportation application where acceleration and deceleration affect the accuracy of sensor values. In [18] the authors proposed a deep-learning neural network (DNN) architecture in which the faulty neurons were tolerated using the Brooks-Iyengar algorithm. To reduce the overhead associated with fault detection, they implemented their architecture on a chip. Brooks-Iyengar algorithm, furthermore, was recently used for blockchain technologies in [19]. Our work applies this algorithm in an IoT application to (i) fuse the values received from redundant sensors connected to each IoT node, and (ii) achieve agreement between the nodes interconnected with pub/sub architecture. The implementation of the algorithm, alongside other components in our methodology, was done using the Discrete Event System Specification (DEVS), a modular and hierarchical formalism. DEVS formalism can be viewed as an extension of finite-state machines, where state transition occurs either due to new input to the system or the expiry of a time variable [20]. The hierarchical nature of DEVS, furthermore, allowed us to create the model of interest using smaller components called atomic models. Each atomic model was responsible for capturing one behavior of the model, and the complete model was created by coupling the atomic models. A previous work that applied a form of multisensor fusion algorithm using the DEVS formalism include [21]. In their work, the authors developed a framework for aiding the implementation of multisensor fusion in DEVS for embedded systems. They also showcased the applicability of their framework by implementing a fusion algorithm based on principal component analysis (PCA). Our work differs from theirs in that it uses DEVS as a formalism to implement the Brooks-Iyengar algorithm, which alongside multisensor fusion, can achieve inexact agreement among the distributed nodes in the network as well. Moreover, we also modeled and simulated the communication of the nodes as well.

The DEVS models in our study were implemented using Cadmium v2 [22], a header-only C++ library. Source code for the Cadmium library and the relevant documentation can be found at *https://github.com/SimulationEverywhere/cadmium_v2*. Cadmium can execute DEVS models for simulation and with a real-time clock, which was then used for their deployment on the hardware. In Cadmium, each DEVS model is created by defining a C++ class derived from the type `Atomic<T>`, where T is a type that represents the state of our model using its declared member variables. The type T can contain primitive data types, as well as user-defined types. The developed atomic models can then be coupled together to construct more complex models. The top-level model built from the atomic and coupled model

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

is simulated by instantiating `RootCoordinator` and calling the `simulated` member function. For real-time execution of the models, instead of `RootCoordinator`, objects of type `RealtimeRootCoordinator` can be instantiated instead.

Our methodology defines a number of nodes that communicate with the publish/subscribe architecture. This paradigm involves entities exchanging messages via a *message broker*, serving as an intermediary. The clients in this paradigm first show their interest in a subset of messages by sending "subscribe" requests to the broker. Later, when a client sends, or "publishes" a message to the broker, the broker forwards the message to any other client that previously showed an interest that matches that message [22]. In a publish/subscribe architecture, the clients can show interest based on the *subject* or *content* of the messages. In a subject-based architecture, the clients publish and receive data from the subjects (also called *topics*) to which the message belongs. In a content-based system, the messages are sent and received based on the information they carry in the body of their messages. The main characteristic of the publish/subscribe architecture, compared to the traditional client-server pattern, is the asynchronous communication and decoupling in both space and time. Space decoupling implies that the parties in communication do not need to hold any references about each other and they only send and receive messages to the broker. Time decoupling refers to the concept that clients can publish messages to the broker regardless of all subscribers being connected to the broker. These messages can then be received by the disconnected clients whenever they establish a connection with the broker again.

MQTT is a lightweight protocol built on top of TCP/IP that implements a publish/subscribe architecture [23]. This protocol was originally designed for resource-constrained devices; however, its features also make it suitable for a broader range of applications. One of these features includes the three levels of quality of service (QoS) for messages, which can guarantee that the message will be received at most, at least, or exactly once. This functionality is implemented using acknowledgment packets which are exchanged after each message. The specification of the MQTT protocol also defines the mechanism to achieve authorization and authentication. A survey [24] compared this protocol with other frequently used protocols and outlined their strengths and weaknesses. Considering these advantages, we decided to adopt this protocol to perform communication between the nodes.

3 SENSOR FUSION AND INEXACT AGREEMENT IN A DEVS-BASED ARCHITECTURE

The developed DEVS models that run on devices are depicted in Figure 1. The figure shows the different DEVS atomic that were defined to read sensor data, fuse, and transmit to other nodes, as well as to achieve inexact agreement among the nodes within the network. The ADC models periodically poll the analog-to-digital channels of the device and capture the values of its sensors. The captured value is the quantization of the analog signals into a digital number that corresponds to the voltage or the current level of the analog input. The quantization of the signal enables the devices to store sensor values in the memory registers and perform the desired computations on them.

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

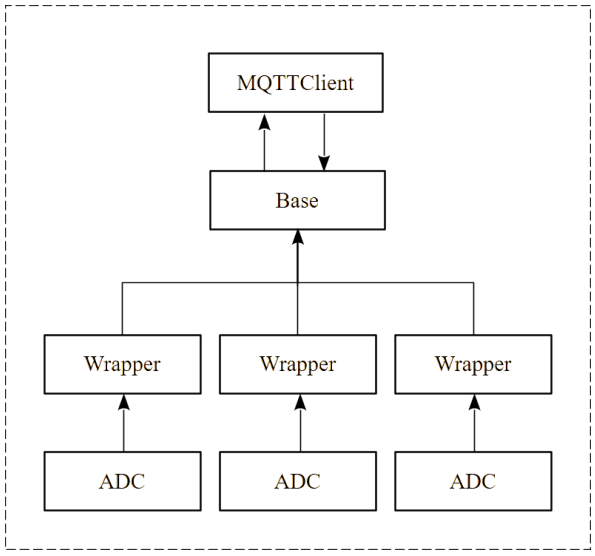

Figure 1: The DEVS models execute on the device.

After the values of sensors are captured by the ADC model, they are then sent to Wrapper models, which are complemented with additional information about the origin of their sensor, as well as a feasible range associated with their values which is required to perform the Brooks-Iyengar algorithm. The information about their origin enables the subsequent models to distinguish the values of an ADC model from the other ADC models. For instance, if we consider an application in home automation, redundant sensors for both capturing CO2 levels as well as temperature might be deployed in the environment. In these cases, the identifiers that are added by the Wrapper models enable the models that receive these values to perform different sets of computations depending on whether the received value is a temperature or CO2 level. Furthermore, the feasible range added by the Wrapper model gives information about the bound in which the actual value of the environment should be present. This range can be determined depending on the physical phenomenon being measured and the precision of the sensors. The advantage of having a separate model, i.e. the Wrapper model, to incorporate additional information is the ability to include the same set of data for multiple ADC models by connecting all of them to the same Wrapper model.

The outputs of multiple redundant sensors, after being complemented by the Wrapper models, are forwarded to the Base model. The Base model was developed to collect multiple sensor readings and compute a final value that would be transmitted from the device. In our context, this calculation is done by applying the Brooks-Iyengar algorithm to the redundant sensors' readings. The output of the algorithm, which is a number and a feasible range of the environment's value, is then sent to the MQTTClient model. The MQTTClient model transmits the values through the network by publishing the data to an MQTT broker. Consequently, any device previously subscribed to the topic that MQTTClient publishes its data will receive these values.

Upon instantiation, the MQTTClient model subscribes to the topics that the 'neighboring nodes' (i.e., nodes that are deployed in the proximity of the node) publish their data to. As a result, devices may receive fused values from neighboring nodes between the intervals of polling their ADCs. When this event occurs, the received data are required to be stored in the memory to later be used for achieving inexact agreement using Brooks-Iyengar's algorithm. This is done by forwarding the data from MQTTClient to the Base model, where they are added to an internal state variable that maps each neighboring node with the value they last reported.

When the content of this "map" becomes full for all neighboring nodes, the Base model (i) applies the Brooks-Iyengar algorithm on the map and saves the result as it is the node's current value (ii) clears the

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

content of the map and (iii) sends the device's current value to the MQTTClient to be broadcasted to other nodes. Each round of the message exchanges between the nodes causes their value to converge closer to each other. By performing these steps, the Base model also increments an internal counter which keeps track of the number of times the internal map becomes full and empty. This counter corresponds to the number of messages the node sends to other nodes between each ADC polling. After the value of the counter reaches a limit, the model will discard any subsequent value that it receives from the neighboring nodes and is not added to the internal map. Furthermore, the choice of the limit for the counter determines the level of accuracy of the node's results, i.e. the larger the limit, the closer the values of the nodes will reach.

In the discussed architecture, if all nodes that are deployed in an environment send and receive data to all other nodes, a new value from the environment can be detected only after a considerable time. This happens since the Brooks-Iyengar algorithm can tolerate up to one-third of "faulty nodes", and a large number of nodes should report a new value of the phenomenon for them to agree on the new value. By limiting the number of nodes that each node communicates with, a subset of nodes can converge independently of other nodes within the network. This is achieved in our architecture by allowing nodes to send and receive data only to the nodes considered their "neighbors".

To implement the communication with neighboring nodes in a publish/subscribe architecture, first, a unique topic for each node is defined which the different MQTTClient models use to publish their data. Next, the MQTTClient model of each node sends a subscribe message for the topics that its neighboring nodes publish their data to. Consequently, whenever a neighboring node publishes data on its topic, its neighbors will receive the values. Furthermore, to determine whether a node is a neighbor of another node, a 2D coordinate is first assigned to each node, and if a node is within Euclidean distance from the 2D coordinate of another node, they are considered "neighbors". Figure 2 depicts the process of identifying neighbors.

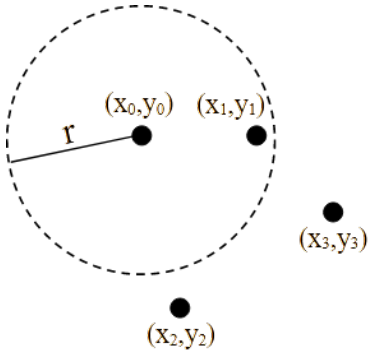

Figure 2: Determining neighboring nodes based on the Euclidean distance of nodes.

4 BROOKS-IYENGAR IN PUBLISH/SUBSCRIBE ARCHITECTURE USING DEVS

The DEVS model described in Section 3 i.e., ADC, Wrapper, Base, and MQTTClient, each contains a set of state variables, functions that change the state of the model from one state to another i.e., the external and internal transition function, and an output function that produces the outputs of the models. To implement these models for both simulation and execution on the devices, the Cadmium library for C++ was chosen. To define DEVS models using this library, the first step was to declare a C++ class that includes a set of member variables that, together, can represent the state of a model at a given time. These member variables could be of primitive types, like `double` and `int` as well as user-defined types that are defined specifically for the program. As an example, this class for the Base model was defined as:

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

```
struct BaseState {

    double sigma;
    Phase phase;
    Message<double> newMessage;
    map<string,double> m;
    int counter; // To keep track of the number of rounds of message exchange
    SensorState()
            : sigma(sigma),
              counter(0) {}
};
```

The member variables declared in this class include member variables that can hold the current time advance variable, a map that holds the reported value of other neighboring nodes, etc. After declaring this type, the atomic model could be defined in Cadmium by declaring a C++ class that is derived from the type `Atomic<T>` where the type `T` is the class that holds the state variables. For the example of the Base model, the DEVS model was created via:

```
class Base : public Atomic<BaseState>
{ /* … */ }
```

By having the class that represents the DEVS model, the next step was to declare the input and output ports for the model. Ports in Cadmium are declared as member variables of type `Port<T>` where `T` should refer to the type of object that they send or receive. For the Base model, the ports were defined as:

```
Port<Message<double>> inData;
Port<MQTTMessage> outData;
Port<MQTTMessage> inData;
```

Lastly, to complete the definition of a DEVS model, four virtual functions had to be overridden. These functions include internal transition, external transition, output, and time advance function which correspond to the definition of the model in the DEVS formalism.

After developing each atomic model individually, the complete model was created by coupling the developed models together. This model is also known as the ‘top-level’ model. In the current version of the Cadmium, the top-level model can be simulated by instantiating the type `RootCoordinator` and executing start() and simulate(*time*) member functions. These steps were performed using the following code:

```
auto model = make_shared<TopLevelModel>("top");
auto rootCoordinator = cadmium::RootCoordinator(model);
auto logger = make_shared<cadmium::CSVLogger>("log.csv", ";");
rootCoordinator.setLogger(logger);
rootCoordinator.start();
rootCoordinator.simulate(std::numeric_limits<double>::
infinity());
rootCoordinator.stop();
```

In contrast, to execute the models with the real-time version of Cadmium, objects of type `realTimeRootCoordinator` were created instead. This was done via:

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

```
auto realTimeRootCoordinator = cadmium::RealTimeRootCoordinator
<cadmium::ChronoClock<std::chrono::steady_clock> > (model, clock);
realTimeRootCoordinator.start();
realTimeRootCoordinator.simulate(std::numeric_limits<double>::infinity());
realTimeRootCoordinator.stop();
```

The result of this compilation is suitable to be flashed and executed on the device. This step can be performed only when the models are tested in a simulation environment and their correct behavior is verified. Simulation in this context, can determine the correct size of neighborhoods by observing the tradeoff between the more resilience against faulty nodes, and the convergence rate that nodes can perform. With realistic models of the environment that would capture how sensors become faulty, the suitable number of redundant components can also be determined using simulation. The Cadmium environment allows execution of the models in simulation mode and then, when satisfied, flash them to the target platform.

5 SIMULATING NODES WITH REDUNDANT SENSORS FOR INEXACT AGREEMENT

We will first discuss a case study featuring 7 interconnected nodes, with each node equipped with 3 redundant sensors. In this setup, the nodes transmit their readings periodically to their neighboring nodes. The assigned location for each node, alongside the nodes that were considered as their neighbors is depicted in Figure 3. The figure shows the nodes and their proximity (with dashed lines) which illustrates other nodes that are considered as the nodes' 'neighbors'. Consequently, all neighboring nodes are subscribed to the topic that the node sends its data to, and they will receive the data whenever the node publishes data. We show the results of two simulations using the DEVS simulator. For both runs, the feasible range of the sensor readings was set to be ±2, and 10 rounds of messages were exchanged between the nodes to reach agreement.

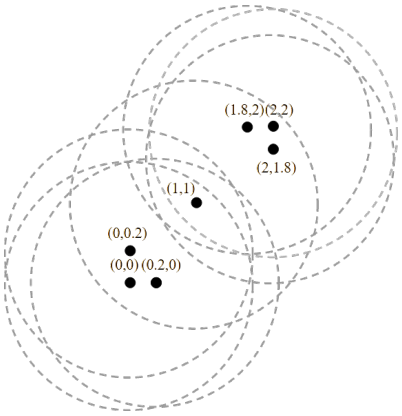

Figure 3: Location and proximity of the nodes in the case study.

As we can see in the figure, the node at location (0,0) was assumed to be faulty and producing values normally distributed with a mean of 8.5 and a standard deviation of 0.2. The rest of the nodes were generating values with a mean of 1.5 and a standard deviation of 0.2. Figure 4 shows fused values that each node reported to its neighboring nodes for each iteration of ADC polling. This figure shows the fused output of four iterations of polling the ADCs for the faulty one (designated with the light blue color) and the rest of the correct nodes.

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

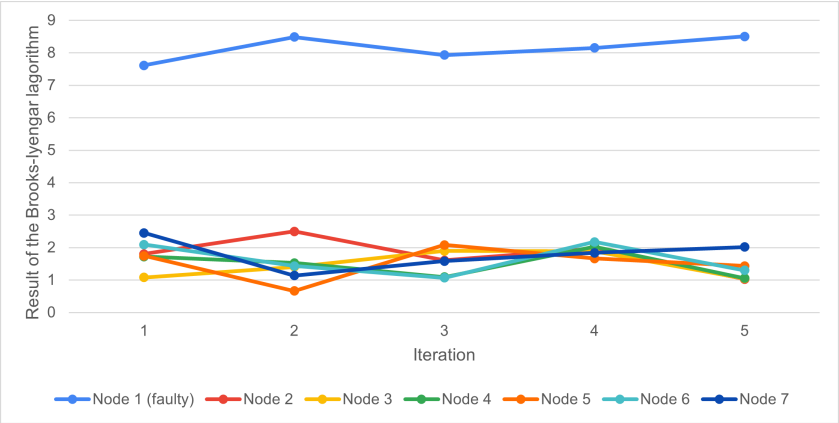

Figure 4: The fused values of each node received from their sensors in simulation #1.

Figure 5 depicts the values of each node after the rounds of messages with their neighbors. The x-axis on the figure shows the number of iterations for which a round of the Brooks-Iyengar algorithm was applied to the content of the internal map of each node. Since, in the simulation run the limit of the counter for this step was chosen to be 10, the nodes' value converges in the interval of every 10 rounds of message exchange. Furthermore, as with the case of practical scenarios, the delay of the network was chosen such that all the rounds of message exchange occurred between a single interval of ADC polling on the devices. As it is evident from the result depicted in the figure, all nodes, including the faulty one, omit the faulty sensor readings and their values converged to common ones. This can be seen from observing the output of the faulty node (designated by the light blue on the y-axis) that on all iterations, starting from the first one, the output is close to the mean value of 1.5 which shows that the faulty value the node initially reported to other ones i.e., the values around the mean value of 8.5 were discarded in its output.

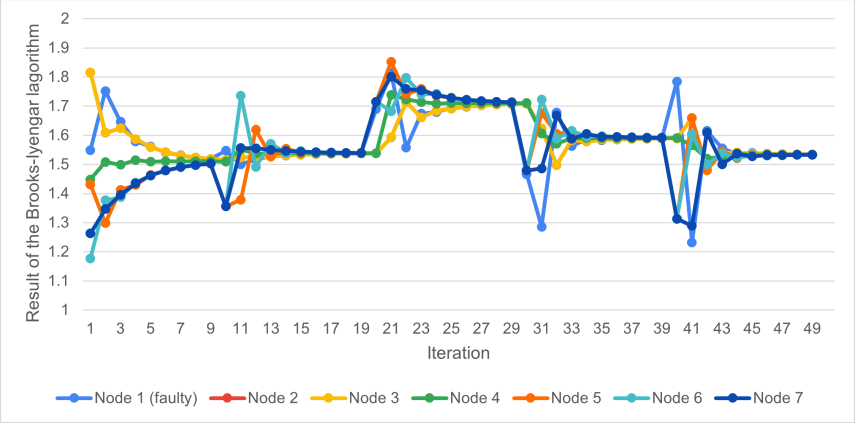

Figure 5: The value of each node after exchanging messages in the first simulation #1.

We now show the results of a simulation scenario in which we observe the effect of different neighborhoods on the convergence of the nodes' values. In this run, the sensors of the three nodes located at (0,0), (0,0.2), and (0.2,0) were all producing values normally distributed with a mean of 8.5 and a standard deviation of 0.2. The rest of the nodes were generating values with the same standard deviation but a mean of 1.5. In this configuration, all nodes were assumed to be working correctly and the difference between the values they reported was due to the actual differences in the environment in which the sensors were deployed. Figure 6 shows the fused value of each node they reported. As the figure shows, the fused value of the redundant sensors of three nodes near the point (0,0) was close to the mean value of 8.5, while for other nodes it was 1.5.

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

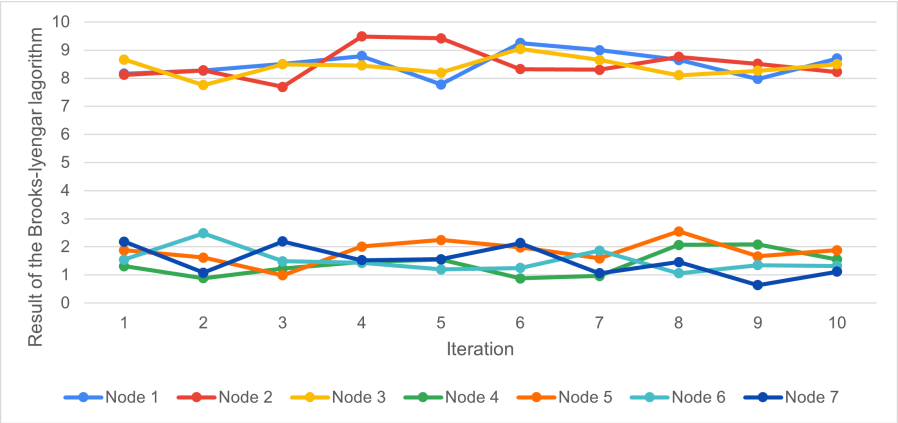

Figure 6: The fused values of each node received from their sensors in simulation #2.

Figure 7 depicts the value of the nodes after the message exchange. As the results indicate, the three nodes that were near the point (0,0), as well as the three nodes that were near (2,2), all converged to common values. Moreover, the node at location (1,1) which was receiving and publishing data to all other 6 nodes, reached the value of approximately 5 which was in-between the two means.

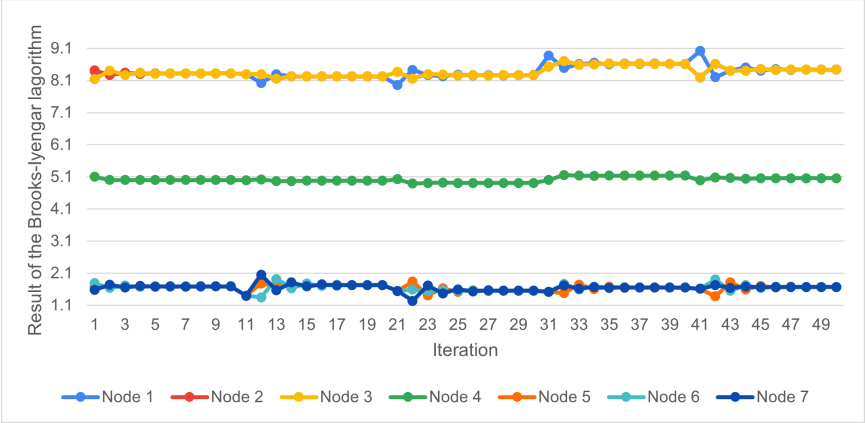

Figure 7: The value of each node after exchanging messages in simulation #2.

The two simulations showed the effect of different neighborhoods and the values of the sensors on the converges of their values. Depending on a particular environment, different sets of simulation runs can be performed to determine the suitable number of nodes, sensors, and neighborhood configuration.

6 CONCLUSION AND FUTURE WORK

A broad range of sensor-based technologies must be designed with the fundamental requirement of accommodating faulty sensors. This paper applied the Brooks-Iyengar algorithm to fuse sensor readings and reach an agreement in the pub/sub architecture. We developed our methodology based on DEVS, a modular and hierarchical formalism, which made the models suitable to execute in simulation and on the devices for their operation.

In a case study, we simulated a network of nodes, each with multiple sensors. The result of the simulation showed the applicability of our approach to dealing with the faulty nodes, and also, the effect it has on nodes' convergence based on their assigned locations.

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

For the future direction of this work, other types of sensor fusion and agreement algorithms can be implemented using the DEVS formalism. As a result, various models of data fusing and processing can be executed in the simulation and the optimal ones depending on constraints and requirements can be chosen.

7 REFERENCES

[1] R. C. Luo, C.-C. Yih, and K. L. Su, "Multisensor fusion and integration: approaches, applications, and future research directions," *IEEE Sensors journal*, vol. 2, no. 2, pp. 107–119, 2002.

[2] S. R. Mahaney and F. B. Schneider, "Inexact agreement: Accuracy, precision, and graceful degradation," presented at the Proceedings of the fourth annual ACM symposium on Principles of distributed computing, 1985, pp. 237–249.

[3] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," in *Concurrency: the works of Leslie Lamport*, 2019, pp. 203–226.

[4] R. R. Brooks and S. S. Iyengar, "Robust distributed computing and sensing algorithm," *Computer*, vol. 29, no. 6, pp. 53–60, 1996.

[5] Y. Labiche and G. Wainer, "Towards the verification and validation of DEVS models," presented at the in Proceedings of 1st Open International Conference on Modeling & Simulation, Citeseer, 2005, pp. 295–305.

[6] H. Saadawi and G. Wainer, "Verification of real-time DEVS models," presented at the Proceedings of the 2009 Spring Simulation Multiconference, Citeseer, 2009, pp. 1–8.

[7] M. M. Olsen and M. S. Raunak, "A method for quantified confidence of DEVS validation.," presented at the SpringSim (TMS-DEVS), 2015, pp. 135–142.

[8] E. F. Nakamura, A. A. Loureiro, and A. C. Frery, "Information fusion for wireless sensor networks: Methods, models, and classifications," *ACM Computing Surveys (CSUR)*, vol. 39, no. 3, pp. 9-es, 2007.

[9] R. Krishnamurthi, A. Kumar, D. Gopinathan, A. Nayyar, and B. Qureshi, "An overview of IoT sensor data processing, fusion, and analysis techniques," *Sensors*, vol. 20, no. 21, p. 6076, 2020.

[10] W. Ding, X. Jing, Z. Yan, and L. T. Yang, "A survey on data fusion in internet of things: Towards secure and privacy-preserving fusion," *Information Fusion*, vol. 51, pp. 129–144, 2019.

[11] F. Alam, R. Mehmood, I. Katib, N. N. Albogami, and A. Albeshri, "Data fusion and IoT for smart ubiquitous environments: A survey," *Ieee Access*, vol. 5, pp. 9533–9554, 2017.

[12] I. Belhajem, Y. Ben Maissa, and A. Tamtaoui, "An improved robust low cost approach for real time vehicle positioning in a smart city," presented at the Industrial Networks and Intelligent Systems: Second International Conference, INISCOM 2016, Leicester, UK, October 31–November 1, 2016, Proceedings 2, Springer, 2017, pp. 77–89.

[13] L. M. Candanedo and V. Feldheim, "Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models," *Energy and Buildings*, vol. 112, pp. 28–39, 2016.

[14] K. Marzullo, "Tolerating failures of continuous-valued sensors," *ACM Transactions on Computer Systems (TOCS)*, vol. 8, no. 4, pp. 284–304, 1990.

[15] S.-C. Wang, K.-Q. Yan, C.-L. Ho, and S.-S. Wang, "The optimal generalized Byzantine agreement in cluster-based wireless sensor networks," *Computer Standards & Interfaces*, vol. 36, no. 5, pp. 821–830, 2014.

[16] S.-C. Wang, S.-C. Tseng, K.-Q. Yan, and Y.-T. Tsai, "Reaching agreement in an integrated fog cloud IoT," *IEEE Access*, vol. 6, pp. 64515–64524, 2018.

[17] B. Ao, "Robust fault tolerant rail door state monitoring systems: applying the Brooks–Iyengar sensing algorithm to transportation applications," *Int. J. Next Gener. Comput*, vol. 8, pp. 108–114, 2017.

[18] P. Sniatala, M. H. Amini, K. G. Boroojeni, P. Sniatala, M. H. Amini, and K. G. Boroojeni, "Designing a Deep-Learning Neural Network Chip to Detect Hardware Errors Using Brooks–Iyengar Algorithm," *Fundamentals of Brooks–Iyengar Distributed Sensing Algorithm: Trends, Advances, and Future Prospects*, pp. 167–174, 2020.

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.

[19] S. S. Iyengar, S. K. Ramani, and B. Ao, "Fusion of the Brooks–Iyengar algorithm and blockchain in decentralization of the data-source," *Journal of Sensor and Actuator Networks*, vol. 8, no. 1, p. 17, 2019.
[20] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of modeling and simulation*. Academic press, 2000.
[21] J. Boi-Ukeme and G. Wainer, "A framework for the extension of DEVS with sensor fusion capabilities," presented at the 2020 Spring Simulation Conference (SpringSim), IEEE, 2020, pp. 1–12.
[22] R. Cárdenas and G. Wainer, "Asymmetric Cell-DEVS models with the Cadmium simulator," *Simulation Modelling Practice and Theory*, vol. 121, p. 102649, 2022.
[23] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.
[24] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.
[25] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," presented at the 2017 IEEE international systems engineering symposium (ISSE), IEEE, 2017, pp. 1–7.

AUTHOR BIOGRAPHIES

IMAN ALAVI FAZEL is a Research Assistant in the Advanced Real-Time Simulation Laboratory, Carleton University, Canada, where he is pursuing a MASc in Electrical and Computer Engineering. His research interests include sensor fusion algorithms and modeling and simulation of IoT technologies. His email address is imanalavifazel@cmail.carleton.ca.

GABRIEL WAINER received the PhD degree from Université d'Aix-Marseille III. He is a full professor in Carleton University. His current research interests are related with modeling methodologies and tools, parallel/distributed simulation, and real-time systems. He is a Fellow of SCS. His email is gwainer@sce.carleton.ca.

Authorized licensed use limited to: Carleton University. Downloaded on November 03,2024 at 17:34:26 UTC from IEEE Xplore. Restrictions apply.