CONTROLLER AREA NETWORK DISCRETE-EVENT SYSTEM SPECIFICATION FOR INDEPENDENT NODE TESTING

Maaz Jamal
Joseph Boi-Ukeme
Gabriel Wainer

Department of Systems and Computer Engineering
Carleton University
Ottawa, ON, CANADA
{maaz.jamal,joseph.boiukeme}@carleton.ca,
gwainer@sce.carleton.ca

ABSTRACT

We introduce a generic method based on the Discrete Event System Specification (DEVS) formalism to model and simulate a Controller Area Network (CAN) port connected to a network with other nodes. The models are tested in software and then implemented in hardware. We show a streamlined development process to model, simulate, and test the use of the CAN network for any general application using the Cadmium simulation environment for software simulations and executing the models on hardware using Real-Time (RT-) DEVS. With this method, different scenarios can be tested in the modeling and simulation phase before the models are ported onto embedded hardware.

Keywords: CAN protocol, DEVS, RT-DEVS.

1 INTRODUCTION

The Controller Area Network (CAN) protocol is used extensively in industrial applications and is now finding use in other applications like automotive, aerospace, and smart buildings. CAN uses a broadcast-based protocol that allows all the nodes to access the packets exchanged on the network. This allows simpler wiring as all the nodes can be connected to a single bus (ISO 2016). Over the years, the applications moved from factory floors to automotive. The number of sensors in a vehicle has increased tremendously and the CAN network has become the de facto standard for vehicle manufacturers to use and allow communication between the nodes in the vehicle (ISO 1993).

The CAN protocol is simple and flexible. It allows the creation of higher-layer protocols that can be tailored to specific use cases. This has led to its adoption in other industries; for instance, UAVCAN, which was designed to be used in Unmanned Aerial Vehicles (UAV) where fault-tolerance and real-time constraints are prioritized (UAVCAN 2021). In addition, the CAN protocol suffers from security issues due to its simple structure. Having a general method to model and simulate CAN ports and nodes connected to the CAN network will allow for easier design and testing of new applications or security features.

Our goal is to advance with new methods for studying CAN nodes, defining a functional communication model for distributed systems and realistic data under application conditions including the number of nodes, communication frequency, error rate, etc. We want to provide reusable models to facilitate the study of a

functional node on a CAN network. Our general approach to modeling the CAN port will allow designers and testers to apply the method to any application that uses CAN. Testing the node response to messages from the network can help us point out potential flaws within the control application.

The CAN architecture supports a modular definition of nodes which allows us to model and test individual nodes and then combine these models to test the final system where we can check if it works as expected. Modeling methods supporting a modular structure, such as the Discrete Event System Specification (DEVS, Zeigler 1989), can help with this task. We used DEVS for modeling the CAN standard and built an RT-DEVS-based CAN port model to demonstrate that nodes can be independently tested to study behavior in a network. The proposed CAN port model allows us to simulate the behavior of a node and test it on a hardware surrogate while it is part of a larger network. The inherent modular nature of DEVS allows this model to be linked to other atomic models and become a part of a larger coupled model. In the final step, models are ported over to hardware using RT-DEVS (Hong et al. 1997).

We focus on modeling at the application layer level (i.e, the physical layer is not modeled), showing how to use our method to design, simulate and test the applications. The model is generic to allow it to be used multiple times including on different nodes in the same network or even the same node as can be the case if a node has multiple CAN ports. By doing this, we SJPW the use of DEVS and RT-DEVS as viable modeling and simulation methodology for CAN that is formally defined, modular, repeatable, and reusable. DEVS can model any application and therefore complete simulation models can be built allowing for extensive testing of a CAN application during production before deployment.

The rest of the paper is organized as follows. Section II describes the background and related work to our proposed methodology and the CAN architecture. In Section III and IV, we discuss the methodology, simulation, and experimental results, and finally, in section V we offer a conclusion.

2 BACKGROUND

The CAN network is a broadcast-based CSMA/RD network. The standard frame has an identifier field that allows the nodes to identify and filter the messages they need. At present, the most widely used version of the CAN protocol is version 2.0 (ISO 2015). The CAN protocol can operate under different data rates (CAN 2.0 has a limit of 1 Mbit/s and a maximum 8 bytes/frame payload), and communication speed is limited by the distance between two nodes (at greater distances lower data rates must be used). CAN Flexible Data-rate (FD) allows for higher bit rates and payloads, and the CAN 2.0 expanded identifier field extends the possible nodes to over 500 million. Simultaneously, this introduces larger overhead per frame, which can be considerable at lower data frame sizes and becomes negligible at larger data frames (Rovira Más, Zhang, and Hansen 2010). The payload for data frames can be between 0-8 bytes (or 0-64 bytes for CAN FD) and a CRC field ensures the integrity of the data.

Different efforts have addressed the CAN protocol in terms of network characteristics and behaviors, security, and expanding its application domain. In (Ziermann et al. 2012) an FPGA-based testbed was developed to analyze the timing behavior of CAN networks to help identify time-critical parameters. In (Ziermann, Salcic, and Teich 2012), a CAN network was specified as a collection of streams from different nodes. For the simulation of the physical layer, (Prodanov, Valle, and Buzas 2009) used analytical methods to design a transceiver model using VHDL-AMS. Their model can be used to design and analyze hardware electrical characteristics. In (Zdeněk and Jiří 2013) the authors designed a spice-based library to allow the study of electrical characteristics of a physical layer of CAN in presence of Electrostatic Discharge.

Owing to the vulnerability of the CAN bus to network security issues (broadcast without authentication), there has been considerable research, including a central authentication scheme to identify nodes securely (Groza et al. 2012; Bella et al. 2019; Groza and Murvay 2018) or using methods for intrusion detection. In (Young et al. 2019) signature-based and anomaly-based methods for intrusion detection are discussed.

CAN has been used in varied domains. In (Ortiz et al. 2011), an HVAC and alarm automation system was implemented using the CAN bus due to its low cost and long range. In (Shweta, Mukesh, and Jagdish 2011) a lighting control system was implemented using the CAN bus as the backbone network.

The CAN bus allows the system designer to choose how to represent data and use built-in arbitration to assign priority to each message. Identifiers are not assigned to CAN ports, but rather to each message. The designer can choose the id for each message, allowing the connection of multiple transducers to a single node and then send out their messages under a different id (Kurachi et al. 2014; Groza et al. 2012).

There are several commercially available tools to simulate CAN networks. CANoe is a widely used tool designed to allow for System Under Test (SUT) in which a hardware or software component can be simulated with test conditions without the risk of damaging components, and CANalyser can be used for studying physical networks (Vector Informatik 2021). These tools are primarily designed for automotive applications, Electronic Control Unit (ECU - the various embedded subsystems found in automobiles), and ECU network simulations. The tool can simulate all the components and their interactions with a CAN network on the datalink and application layers. CANoe has been used to build a test environment for vehicle body simulations (Zhou, Li, and Hou 2008), to test for vulnerabilities by hacking into it (Pimple 2018), and, using CARLA (an open-source simulator based on Unreal Engine), to detect intrusions in the network for automotive applications (Casillo et al. 2019). These tools have limitations; for example, CANalyser does not simulate a CAN network and is only used to stimulate and analyze the physical ECU CAN network for diagnostics purposes. As CANoe is heavily focused on automotive applications, adapting the software for use in other areas will require making custom Functional Mockup Units (FMU) based on Functional Mockup Interface (FMI). Both CANoe and CANalyzer are commercial products.

Various research efforts used formal methods to improve the testing of different aspects of CAN. As an example, (Wang et al. 2020) proposed to model a gateway using timed automata. Their purpose was to test the correctness of message transmission at various rates. The real-time characteristics of the models were verified using Uppaal, a model checker used to verify timed automata. Although formal methods set the foundation to test more advanced scenarios, most methods in the literature have limited use cases and cannot be applied to design a working system relying on a CAN bus with multiple nodes. This raises the question of the potential for other formal techniques to create full-fledged applications using the CAN network. One method that has been applied to similar protocols is RT-DEVS, based on the DEVS formalism. DEVS breaks down complex systems into atomic and coupled models where atomic model specifies the behavior and coupled model specifies the structure. DEVS provides a rich structural representation of the components and allows to explicitly specify timing, making it easy to adapt to real-time systems (Wainer 2009). RT-CADMIUM (Belloli et al. 2019; Earle et al. 2020) implements RT-DEVS on Linux as well as bare hardware using the Mbed library. The tool provides a simple workflow where the models can be defined in DEVS, and the tool will compile and flash the models to firmware. Mbed supports a wide range of hardware, and the applications can be thus tailored to a variety of scenarios. RT-CADMIUM can also encapsulate Mbed native drivers for protocols. This then allows the model designer to let Mbed take care of low-level details of the protocol (acknowledgments, arbitration), and allows them to focus on modeling sending, and receiving messages. RT-CADMIUM and Mbed are open-source libraries.

To adapt DEVS to simulating a CAN port, coupled models can be compiled to be flashed onto hardware nodes where we can run tests individually and combined in a bigger network to check for proper working. Each node can be tested individually and independently from the network while still and we can be reasonably certain that it will function as desired when added to a network. For testing on hardware, we designed the CAN network using a CAN controller and a CAN transceiver IC. The transceiver IC converts the digital serial inputs from the CAN controller into physical voltage levels. We used an MCP2551 transceiver IC as it has low RFI emissions and high noise immunity while supporting up to 1 Mbit/s communication. The CAN controller is a NUCLEO-F207ZG board running Mbed. The board contains two CAN ports and large ROM and SRAM allowing complex models to be flashed onto the system.

3 DEFINING THE CAN PORT

By implementing the CAN port on the application level and using RT-DEVS and using the CADMIUM tool we can design complete applications and simulate them on both software and hardware. The models can be flashed unto hardware as is and therefore we have parity between models.

To test the models, we use three hardware setups that demonstrate that this method of modeling CAN port can result in creating a fully functioning system that will work when connected in a bigger setup. Each of the setups tests a specific use of the CAN port and comprise of testing the ability of a node to send CAN message to the network, receive CAN message, and both send and receive CAN messages. The ability to both receive and send messages while polling a line introduces some complications that we will discuss and offer a solution in this section. The controller is coupled in slightly different ways depending on the simulation type, whether it's the hardware or software simulation.

3.1 CAN Controller Model

The CAN controller Model is responsible for reading messages from the CAN network and passing the messages to the node. The model is defined as a DEVS atomic model. A reduced formal definition of the atomic model that includes the input/output ports and states is presented as follows:

$$
\begin{aligned}
&Can\ Controller\ =<\ X, Y, S, \delta\_int, \delta\_ext, \lambda.ta> \\
&X\ =\ \{\ Rx, DataIn\ \} \\
&Y = \{\ Tx, DataOut\} \\
&S = \{\ state\ \epsilon\ \{transmit, recieve, mode, internaldone\}\}
\end{aligned}
$$

The model uses two input and output ports. The input port 'Rx' is used to receive the messages from the CAN network and 'DataIn' is used to receive inputs from the node to be later transmitted to the network. Similarly, the output port 'Tx' is used to send messages to the CAN network and 'DataOut' is used to send decoded messages to the node from the CAN network. The first two states 'transmit' and 'receive' determine whether the model needs to send a CAN message to the network or has received a CAN message from the network respectively. It should be noted that both 'transmit' and 'receive' can be true at the same time. In this case, the output function will send a CAN message to the network and a message to the node.

The state 'mode' determines the function of the CAN controller in this node. The three modes are transmit-only, read-only, and read-transmit. The transmit-only mode reads inputs from the node through the 'DataIn' port and then sends the received messages through the 'Tx' port in the form of a CANMessage. Since the port is only supposed to transmit messages, the model can be passivated after the CANMessage is sent. The controller then waits until an input is received through the 'DataIn' port to send another CAN message. The read-only mode waits for the next input from the network through the 'Rx' port by continuously polling the line at fixed intervals. If an input is received the controller decodes the received message, which is then sent to the node through the 'DataOut' port. Finally, the read-transmit mode both reads from the network and transmits messages to the network. As before the transmitting data works by the triggering of an external transition through the 'DataIn' port which sets the 'transmit' state to true and then transmits the data to the CAN network. The controller polls the network for new messages; when it receives a message, it transfers the message using the 'DataOut' port and sets the 'internaldone' state to false, passivates the controller and waits (for the node to process the data it has received and prepare any data it wants to send). Polling the line takes computing time away from the rest of the node and the simulation proceeds slowly or not at all depending on the polling rate. By passivating the controller, the rest of the node can have all the computing resources quickly process the data and generate an output for the controller to send. The 'internaldone' state then specifies if the node has completed its processing for a received CANMessage. The model converts the message from type CANMessage as defined in Mbed CAN API to a user-defined internal message. In our case, the user internal message retains the ID and data of the message and strips the remaining components. The ID is retained so that the models connected to the CAN controller can filter

the messages. The modeler can specify a custom internal message that retains more or less of the received CANMessage. It is important to note the CAN controller does not set the ID of the message. Neither does it strip ID information from the message. This allows the other atomic models to decide what ID they will use and how will they react to different ID messages and the information contained in them.

We couple the controller in different arrangements depending on if the simulation is to be done in software or on hardware. For a software simulation, we use the coupling as shown in Figure 1.

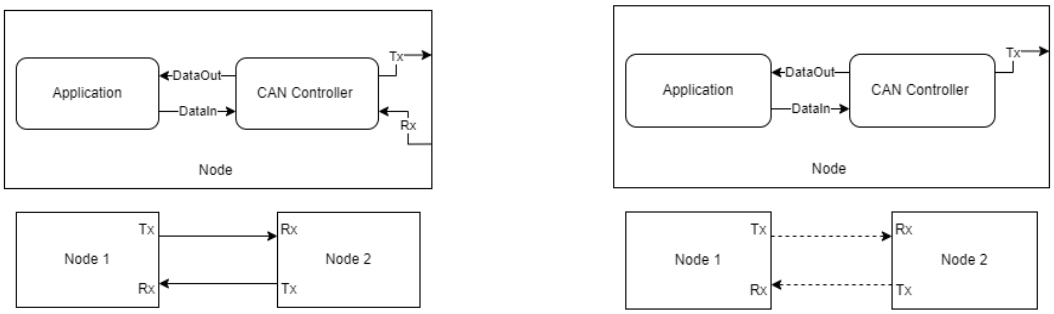

Figure 1: Coupling (a) for software simulation (b) Hardware deployment

In Figure 1a, the top model shows the coupling of the CAN controller atomic with the rest of the 'Application' and the coupled model Node. The controller has an External Input Coupling with 'Tx' port and an External Output Coupling with the 'Rx' port. The 'Application' here represents any coupled or atomic model the node is supposed to do and it communicates with the CAN controller and by extension the network using the Internal Couplings (ICs) with the 'DataOut' and 'DataIn' port of the controller. Figure 1b shows the couplings when deployed in hardware. The only difference is the missing 'Rx' port. This is because reading the network is done continuously by setting the time advance to zero and using the internal transition function to read the network. We poll the network using the internal transition, and the model still uses the 'Rx' model of the physical CAN port, otherwise, it would not be able to receive messages from the network. The coupled model on the bottom shows two nodes, 'Node 1' and 'Node 2' connected by coupling the 'Tx' port of one with the 'Rx' port of the other and vice versa. We can connect multiple Nodes this way. In hardware, the lines connecting the models are dashed, because the nodes are connected over a physical CAN network and are not directly coupled. Each node is a coupled model on a separate physical board connected to the network. We can introduce a separate model between the nodes, this is useful in cases where we want to test the system under conditions like delay or packet loss. In this paper, we focus on demonstrating the use of DEVS and RT-DEVS for simulating the CAN port.

3.2 Hardware Experimental Setup

We built an experimental setup with three boards to form nodes and exchange information between them. The three boards represent a console unit, a motor unit, and the accelerator and brake unit. The nodes are connected over the CAN network. The boards have two pins (TX and RX) for the CAN port. TX is used to send data to the CAN transceiver; RX receives data from the transceiver. Both pins are digital outputs and serial. TX and RX are connected to the respective pins on the MCP2551 IC. The two wires leaving the IC are CANH and CANL wires and together they carry each message; they have differential voltages applied to them so even though they may be two wires they still carry data serially. The three nodes are connected on a signal bus and therefore receive all communications on the network. This method can still be used to model situations where a node is connected to two or more CAN networks that are separated. For example, we can create a scenario where the network is divided into two networks: one for infotainment and climate control and the other for critical components. We can use one board with its two ports to connect to the different busses and use it without changing the port models. The complete system is shown in Figure 2. The system is made up of the Console Unit, the Motor unit, and the Accelerator & Brake unit.

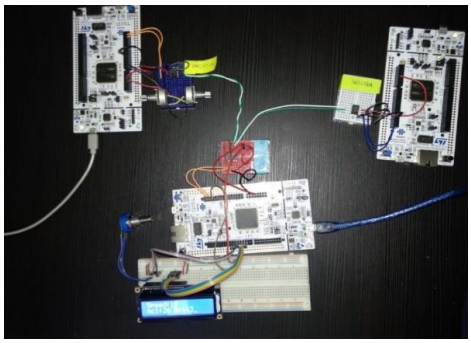

Figure 2. Three nodes and connections between them

The *console unit* is used to process and display the information being passed through the CAN bus from the other units. The *motor unit* simulates a motor operating in a vehicle and simulates the effects of acceleration, drag, and braking forces. The *accelerator and brake unit* converts analog inputs to CAN messages and passes them on to the motor unit. It takes analog inputs from analog pins and can be changed using a potentiometer. The inputs have separate ids and are sent over the CAN network, allowing the motor unit to receive and at the same time distinguish between the two inputs. The motor unit uses this information to compute the speed and send this information back over the CAN network.

4 SIMULATING THE CAN NETWORK

In this section, we discuss the simulation scenarios and test results of the simulations run in CADMIUM and real-time execution on RT-CADMIUM. The tests simulate three units of an automobile and each unit tests one aspect of the CAN port as discussed in the previous section. The three units are the Acceleration & Brake Unit, the Motor Unit, and the Console Unit. The Acceleration and Brake Unit test the ability to send messages over the network. The Brake Unit converts inputs to CANMessages and passes them onto the CAN network without receiving anything in return. The Motor unit tests the ability to update the parameters of the Motor unit based on inputs from the network consisting of acceleration and braking. The motor also sends the speed of the vehicle to the network in periodic intervals. The Console unit receives messages from the other nodes about the current braking, acceleration, and motor speed from the network and displays it to the user. In this manner, each of the hardware setups tests all three of the scenarios a CAN port and a node should be expected to handle effectively and correctly.

4.1 Experimental Setup

Figure 3 shows the couplings of the experimental setup, including the couplings of the three units. The Accelerator Brake CAN coupled model takes analog inputs and passes them into the network using the CAN controller. The inputs are taken from analog pins of the board using ‘AnalogIn’ model and passed onto ‘float_to_int’ model that converts the floating-point values to unsigned bytes as required by the CAN controller and adds id to the message. The Console Coupled model displays the values of the motors' speed, acceleration input, and brake input as received from the CAN network using the ‘Display’ atomic model. In the hardware setup, the display is an LCD connected to the board. The Display atomic model acts as a wrapper for the TextLCD library. The model deciphers the message source from the ids. The text is preformatted and only the values of the relevant fields are changed to express an update. Finally, the Motor Coupled model receives acceleration and braking input from the CAN network and sends speed information to the network. To simulate motion, the model needs to increase its velocity in increments when an accelerating input is given. Slow down when braking or decelerating input is given. Removing the accelerating input should cause the velocity to decrease over time and reach zero after a while. We, therefore, need an iterative equation.

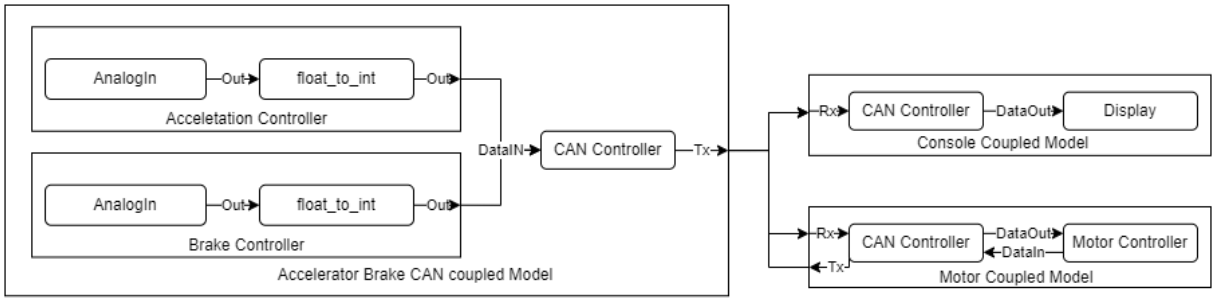

Figure 3: Coupled model showing the couplings of the experimental setup for the three units.

When a vehicle is in motion it is affected by a drag force due to the presence of air resistance. We use the drag force equation to model air resistance, R = 0.5*p*v^2*Cd*A where the symbols p, v, Cd, and A stand for the density of the fluid, velocity of the object, drag coefficient, and cross-sectional area, respectively. In this model, we simplify the drag equation by combining it into one term and keeping the velocity of the vehicle as the only variable. The drag force then becomes 0.5*v^2*w where w is the drag weight. We add the acceleration and braking inputs to the equations by considering them as weighted forces applied to the vehicle, Acceleration is added, while braking force is subtracted. The final iterative equation becomes v(t+1) = v(t) + (w1*a – w2*b- v^2*w). The weights have been set to w1 = 0.25, w2 = 0.5 and w = 0.01. The accelerating input is weighted such that more acceleration is needed to overcome the force of friction generated by the brakes. The drag coefficient is dependent on the previous velocity. A constant velocity is achieved over time as drag force equals the acceleration force.

4.2 Simulation Scenarios

Each node was individually tested; however, for the sake of brevity, we show the final simulation scenarios where the nodes are connected as shown in Figure 3. The scenarios involve sending an accelerating input to the motor, followed by applying a braking input, then removing the braking input, and finally removing the accelerating input. The purpose is to show that the inputs are transferred across the network and the motor acts as we designed (that is, the motor speeds up incrementally if an accelerating input is applied, slows down incrementally if a decelerating input is applied, and slows down if an accelerating input is removed). Table 1 shows the output from the acceleration unit when it is increased from minimum to max. This scenario also tests the *float_to_internal* atomic models as they are used in the simulation as well. The model receives inputs from AnalogIn and passes them on to the *float_to_internal* atomic model which passes them out to the CAN atomic model and then the CAN bus. Table 1 shows some of the logs when the analog inputs are increased from 0.0 to 1.0 in increments of 0.1.

Table 1. Simulation output

Analog Input	**Float_to_internal**		**CAN1 TX output**	
	ID	**Data**	**ID**	**Data**
0	1100	0	1100	0
0.1	1100	25	1100	25
0.5	1100	127	1100	127
0.9	1100	229	1100	229
1.0	1100	255	1100	255

As discussed in Section 4.1, the analog inputs are converted, by the *float_to_internal* into bytes and add an id to the message. This is passed to the CAN controller model using the ‘DataIn’ port, where it is sent

through the 'Tx' output port to the network. This input reaches the motor unit, and we log the data received at the input and the resulting speed of the motor. The graph in Figure 4a, shows the speed of the motor as a result of inputs from the acceleration and braking units.

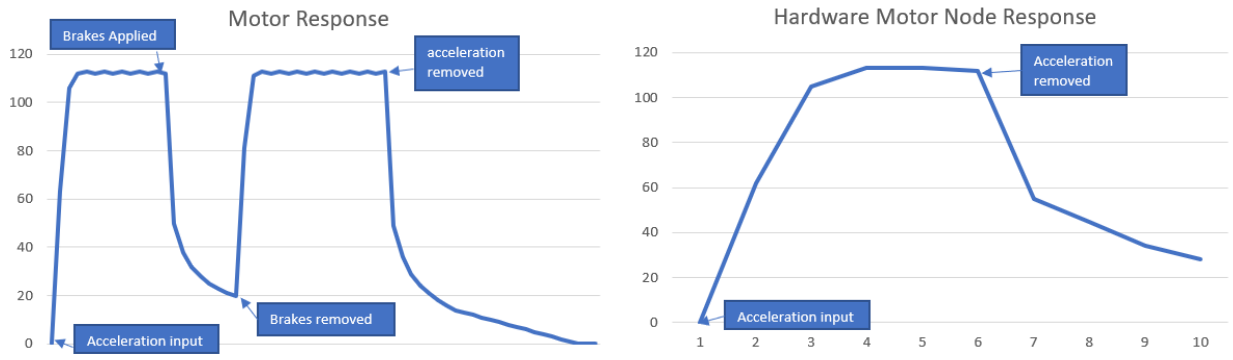

Figure 4. Motor speed output graph. (a) software simulation (b) hardware exection.

We can see that when the accelerating input is applied, the motor reaches maximum speed. When braking is applied, the speed falls incrementally. We then remove the braking input and see that the speed increases back to its top speed. Finally, the accelerating input is removed, and we can see that the speed falls due to drag force until eventually reaching zero.

If we analyze the simulation log files, we can see that the motor receives the input, calculates the speed, and then return the message to the CAN controller for further transmission on the CAN bus. Since the CAN controller is in mode 2 i.e read-transmit the CAN controller receives a message and then stops transmitting until the other model returns data through the DataIn port, in this case, the motor atomic model. The first few outputs from the logs are shown in Table 2 with extra information removed.

Table 2. Selected motor log outputs

Input		**CAN DataOUT**		**Motor Output**		**CAN TX**	
ID	**Data**	**ID**	**Data**	**ID**	**Data**	**ID**	**Data**
1200	255	1200	255	1300	0	1300	0
				1300	63	1300	63

The ID 1200 represents an accelerating input. The ID 1300 is reserved for the Motor data representing speed. The input for acceleration is applied once and then received by the CAN port, the motor upon receiving the input starts calculating its speed iteratively. Hence why there is only one entry for the Input and CAN DataOut and multiple entries for Motor Output and CAN TX.

4.3 Deployment on the target hardware platform

The hardware execution was carried out using the setup shown in Figure 2. The difference is as discussed in Section 3.1 each node is on a separate embedded board with its own CAN port and connected to a real network. We record the log outputs over serial USB into a computer. Three log files were obtained from each of the nodes: Accelerator and Brake unit, Console/Display unit, and Motor unit. Analyzing the log file from the accelerator unit we can see the log shows the input from the two analog pins and the outputs from the *float_to_internal* and CAN port. However, we do not see any input from the CAN port, as it should be, as this node is in transmit-only mode. The log for the CAN port message does not show any values as in this case it uses the Mbed OS representation of CANMessage and the logger cannot decipher the message. Table 3 shows some of the outputs from this node. Table 3 represents twisting the potentiometer to a

minimum resistance for the accelerator pin and then doing the same for the brake analog potentiometer. The analog input shows the intensity of analog input at the pin while the other column shows the converted values. If we look at entries 4 and 5, we can see that 0.98 gives us 25, which is approximately 1/10 of 1.0 and 255. In the same manner, the 0.610 is converted to 155 which is approximately 60% of 1.0 and 255 so we can conclude this node is working as intended.

Table 3. Logged data from Accelerator and Brake node.

Analog Input Acceleration	Acceleration Output		Analog Input Brake	Brake Output	
	ID	Data		ID	Data
0.165079	1200	42	0	1100	0
1	1200	255	0.0981685	1100	25
1	1200	255	0.999512	1100	254

Moving onto the motor node log files (a small, selected subset is shown in Table 4). We can see the inputs from the accelerator and brake node are successfully reaching the node over the CAN bus. The motor is calculating the speed and then transmits the values over the CAN bus.

Table 4. Log output of the motor node.

Motor model Output		**CAN DataOut**	
ID	**Data**	**ID**	**Data**
1300	0	1200	254
1300	62	1100	0
1300	105	1200	254
1300	112	1200	23
1300	45	1200	0

We can see the CAN port is receiving both accelerating and braking inputs. In this subset, the braking input is set to zero while the accelerating input is set to max and then set to zero again. The motor calculates this speed and sends it back to the CAN port which transmits it with ID 1300. Due to limitations of the buffer size of the logging software a complete log of the output of the motor was not achieved. However, if we plot the data that we have, as shown in Figure 4b. We can see the plot resembles what we obtained in Figure 4a for maximum acceleration and then removing the accelerating input.

Finally, we move to the console unit logs. This node receives inputs from the other two nodes and tells us that the network is working properly. A small subset of the logs is displayed in Table 5. We can see the console node is receiving all the outputs from the other two nodes and hence we can conclude that the network is working properly.

Table 5. Console log.

CAN DataOUT	
ID	**Data**
1300	66
1200	154
1100	24

In the results discussed in this section, we can see how we first tested the Acceleration and Brake unit, and the Motor unit in software to check for proper working. The testing used pseudo values for acceleration and

braking and showed them being relayed onto the simulated network. This showed that the models used to model the acceleration and brake unit were outputting information as we designed and with the correct id.

The motor testing provided us with the result that the model that we designed is correctly receiving information from a simulated network and is then computing and sending data back to the network. After the nodes are tested individually and in the complete model we test on hardware. The final testing on the hardware with all the components connected in a network agreed with our results from the software simulation. We can then reasonably be confident that our method and model of simulating a CAN port can be used to rigorously test nodes individually and be certain that they will work in a larger hardware setup.

The logs for the software simulation can allow for the establishment of a temporal connection between messages in the log as each of the nodes is simulated as a complete coupled model and therefore the logs follow a global time showing causation more simply. In hardware, each of the nodes is an independent coupled model on its own with its own local time for the logs. This makes it impossible to use the logs to establish causation between events happening in one log and the events occurring in another.

These discrepancies however do not affect our results significantly. Although we cannot establish a temporal connection between two logs we can, however, compare the trends shown in both logs and use them to confirm or deny our proposition. In our results, we found despite the smaller log files and different log entry times that the two simulations in software and hardware follow the same pattern shown in the graphs for the speed. The same pattern allows us to conclude that the models are behaving similarly in both scenarios and our methodology is sound.

The motor model iteratively computes the speed by considering the acceleration, braking, and drag force on the vehicle. The model can be improved by adding the effect of gears on top speed by changing the coefficients of the model. In the same vein, a more accurate equation for the drag force and its coefficients can more closely model a vehicle in motion

5 CONCLUSION

In this paper, we defined a CAN controller using DEVS formalism in software and then simulated the CAN controller on RT-DEVS. The models were implemented on NUCLEO-F207Z boards running Mbed and CADMIUM. Implementing the CAN bus now allows us to simulate boards independently and still allow communication between the boards on actual hardware. The results show us that we can use DEVS and RT-DEVS to accurately model a CAN port and this allows us to independently model and test nodes in software before implementation in hardware. The confirmation of the results in hardware opens a viable alternative method to designing and testing systems that use a CAN network for communication.

The method introduced here can be generalized and be applied to other scenarios. The CAN port was kept generic to allow it to be used in any application. The CAN port can be reconfigured, and different data types sent over the CAN port provided an appropriate converter to CAN frame is modeled and connected between the application and CAN port. Other higher layer protocols can also be used using this methodology by using appropriate models to send data to the CAN port.

We have demonstrated the CAN port can send, receive, or do both on nodes with our testing and that this functionality can implement many different scenarios. As an example, the vehicle model can be replaced with a building model. In this model the console can turn into a display unit, the acceleration and brake unit can be converted to sensors reporting environmental data to a control unit that receives and then sends data back to the network to actuators on the CAN network. We can also simulate a higher layer protocol like CANopen.

The models can be further improved by solving some issues. The current method of polling the line prevents the model from proceeding and makes the other model dependent on the input from an external source to

proceed with their simulation. Using an interrupt-based model that uses interrupt signals from the internal registers can allow the rest of the models on a node to proceed with their simulation and not be reliant on receiving a message from the node. Implementing a higher layer protocol like CANopen using this methodology and testing will allow further confidence to be built in the use of RT-DEVS for testing and implementation purposes. Furthermore, it will open more applications that rely on CANopen to be designed and tested using this methodology. In our testing, we found no significant deviation from simulation in software and testing on the hardware of our example of a car motor, acceleration, and braking system.

REFERENCES

Bella, G., P. Biondi, G. Costantino, and I. Matteucci. 2019. "TOUCAN: A proTocol tO secUre Controller Area Network". *Proceedings of the ACM Workshop on Automotive Cybersecurity*, pp. 3-8. https://doi.org/10.1145/3309171.3309175.

Belloli, L., D. Vicino, C. Ruiz-Martin, and G. Wainer. 2019. "Building Devs Models with the Cadmium Tool.". In *2019 Winter Simulation Conference (WSC)*, pp. 45–59. National Harbor, MD.

Casillo, M., S. Coppola, M. De Santo, F. Pascale, and E. Santonicola. 2019. "Embedded Intrusion Detection System for Detecting Attacks over CAN-BUS.". In *4th International Conference on System Reliability and Safety (ICSRS 2019)*, pp. 136–41. Rome, Italy.

Earle, B., K. Bjornson, C. Ruiz-Martin, and G. Wainer. 2020. "Development of a Real-Time Devs Kernel: Rt-Cadmium.". In *Spring Simulation Conference (SpringSim 2020)*, pp. 1–12. Fairfax, VA.

Groza, B., and P.-S. Murvay. 2018. "Security Solutions for the Controller Area Network: Bringing Authentication to In-Vehicle Networks.". *IEEE Vehicular Technology Magazine* vol. 13, no. 1, pp. 40–47.

Groza, B., S. Murvay, A. van Herrewege, and I. Verbauwhede. 2012. "LiBrA-CAN: A Lightweight Broadcast Authentication Protocol for Controller Area Networks.". *Cryptology and Network Security* vol. 7712, pp. 185–200. https://doi.org/10.1007/978-3-642-35404-5_15.

Hong, J. S., H. S. 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* vol. 7, no. 4, pp. 355–75. https://doi.org/10.1023/a:1008262409521.

International Organization for Standardization. 1993. ISO 11898:1993. Geneva, Switzerland: International Organization for Standardization (ISO).

———. 2015. ISO 11898-1:2015. Geneva, Switzerland: International Organization for Standardization (ISO). https://www.iso.org/standard/63648.html.

———. 2016. ISO 15765-2:2016 Road Vehicles — Diagnostic Communication over Controller Area Network (DoCAN). Geneva, Switzerland: International Organization for Standardization (ISO).

Kurachi, R., Y. Matsubara, H. Takada, N. Adachi, Y. Miyashita, and S. Horihata. 2014. "CaCAN-Centralized Authentication System in CAN (Controller Area Network).". In *Embedded Security in Cars (ESCAR)*. Hamburg, Germany.

Ortiz, M., M. Diaz, F. Bellido, E. Saez, and F. Quiles. 2011. "Smart Home Automation Using Controller Area Network.". *Advances in Intelligent and Soft Computing* vol. 91, pp. 167–74.

Pimple, P. 2018. "Sniffing the Automotive CAN Bus for Real-Time Data-Logging and Real Time Diagnostics Display." In *Intl Conference on Smart Electric Drives and Power System*. Nagpur, India.

Prodanov, W., M. Valle, and R. Buzas. 2009. "A Controller Area Network Bus Transceiver Behavioral Model for Network Design and Simulation.". *IEEE Trans. on Industrial Electronics* vol. 56, no. 9, pp. 3762–71.

Rovira Más, F., Q. Zhang, and A. C. Hansen. 2010. "Communication Systems for Intelligent Off-Road Vehicles." In *Mechatronics and Intelligent Systems for Off-Road Vehicles*, pp. 187–207. London, Springer. https://doi.org/10.1007/978-1-84996-468-5_6.

Shweta, S. A., D. P. Mukesh, and B. N. Jagdish. 2011. "Implementation of Controller Area Network (CAN) Bus (Building Automation)." In *Communications in Computer and Information Science* vol. 125, pp. 507–514. Berlin, Heidelberg., Springer. https://doi.org/10.1007/978-3-642-18440-6_64.

UAVCAN. 2021. "UAVCAN (Uncomplicated Application-Layer Vehicular Computing and Networking)". UAVCAN organization. April 1, 2021. https://uavcan.org/.

Vector Informatik. 2021. "CANoe | ECU & Network Testing | Vector.". www.vector.com. https://www.vector.com/ca/en/products/products-a-z/software/canoe/.

Wainer, G. 2009. *Discrete-Event Modeling and Simulation*. Boca Raton, FL, USA: CRC Press.

Wang, R., Yong G., X. Li, and R. Zhang. 2020. "Formal Verification of CAN Bus in Cyber Physical System.". In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion* (QRS-C), pp. 249–55. Macau, China.

Young, C., J. Zambreno, H. Olufowobi, and G. Bloom. 2019. "Survey of Automotive Controller Area Network Intrusion Detection Systems.". *IEEE Design & Test* vol. 36, no. 6, pp. 48–55.

Zdeněk, K., and S. Jiří. 2013. "Simulation of CAN Bus Physical Layer Using SPICE.". In *2013 International Conference on Applied Electronics*, pp. 1–4. Pilsen, Czech Republic.

Zeigler, B. P. 1989. "DEVS Representation of Dynamical Systems: Event-Based Intelligent Control.". *Proceedings of the IEEE* vol. 77, no. 1, pp. 72–80. https://doi.org/10.1109/5.21071.

Zhou, F., S. Li, and X. Hou. 2008. "Development Method of Simulation and Test System for Vehicle Body CAN Bus Based on CANoe.". In *2008 7th World Congress on Intelligent Control and Automation*, , pp. 7515–7519. Chongqing, China. https://doi.org/10.1109/WCICA.2008.4594092.

Ziermann, T., A. Butiu, J. Teich, and D. Ziener. 2012. "FPGA-Based Testbed for Timing Behavior Evaluation of the Controller Area Network (CAN)." In *2012 International Conference on Reconfigurable Computing and FPGAs*, pp. 1-6, Cancun, Mexico.

Ziermann, T., Z. Salcic, and J. Teich. 2012. "Improving Performance of Controller Area Network (CAN) by Adaptive Message Scheduling." In *Self-Organization in Embedded Real-Time Systems*, edited by M. Higuera-Toledano, U. Brinkschulte, A. Rettberg, pp. 93–118. New York.

AUTHOR BIOGRAPHIES

MAAZ JAMAL is an M.A.Sc student at Carleton University. His research interests lie in embedded systems and real-time simulations. His email address is maaz.jamal@carleton.ca.

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 SCS. His email address is gwainer@sce.carleton.ca.