Applying Modelling and Simulation for Development of Embedded Systems

Gabriel Wainer
Joseph Boi-Ukeme

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

ABSTRACT

Formal methods and tools help in building embedded systems with real-time constraints, but most existing methods are still hard to scale up. Instead, systems modeling and simulation (M&S) can improve the development task and provide higher quality. M&S is widely used for the early stages of projects; however, when the development tasks switch towards the target environment, early models are often abandoned. In order to deal with these issues, we introduced a methodology based on discrete-event systems specifications, which combines the advantages of a practical approach with the rigor of a formal method, in which one consistently use models throughout the development cycle.

Keywords: Embedded Systems, Model Driven Development, DEVS.

1 INTRODUCTION

Formal methods for embedded systems development use mathematical notations to define the system's requirements, allowing proving system properties (liveness, timeliness, etc.). These techniques have had success, but they are still difficult to apply and do not scale up well. Instead, construction of system models and their analysis through simulation (M&S) reduces cost and risk, allowing exploring changes and testing of dynamic conditions in a risk-free environment. This is a useful approach, moreover considering that testing under actual operating conditions may be impractical and in some cases impossible. Our research focuses on how to bridge formal methods and formal M&S for to analyze RTS and for studying their interaction with the physical environment, while reusing the original models. The methodology, called DEMES (for Discrete-Event Modeling of Embedded Systems), is based on DEVS, as it provides an abstract and intuitive way of modeling, independent of underlying simulators, hardware, and middleware (Zeigler et al. 2000). DEMES combines the advantages of a practical approach with the rigor of a formal method, in which one consistently uses the same models throughout the development cycle, and it has been presented in (Niyonkuru and Wainer 2015; Niyonkuru and Wainer 2016; Saadawi and Wainer 2009; Saadawi and Wainer 2010). DEMES enables the incremental construction of such embedded applications using a discrete-event architecture for both simulation and the target. The use of DEVS for DEMES offers the following advantages:

- Reliability: logical and timing correctness rely on DEVS mathematical theory.

- Model reuse: DEVS has well-defined concepts for coupling of components and hierarchical, modular model composition.

- Modeling: many techniques used for embedded systems (e.g., SystemC, TA, State Charts, etc.) have been mapped into DEVS. We can use different methods while keeping independence at the level of the executive, using the most adequate technique on each part of system architecture and reusing existing expertise.

- Hybrid modelling: the theory includes methods to define the environment under control using continuous modeling methods and approximations.

In the following sections we discuss the DEMES methodology, related work and tool. We will introduce a tutorial case study based on a Railway control system, using DEMES for the development process owing to its unique advantages highlighted above.

2 BACKGROUND

DEMES uses M&S and formal methods as an alternative method for embedded software development. The application of M&S for real-time system design has gained popularity in recent years because construction of the system model and analysis by simulation enhances both the capabilities of the system and improves the quality of the final product while reducing cost and risks (Wainer 2015).

DEVS is a timed event system specification for modeling and analyzing discrete event dynamic systems. A real system modeled using DEVS is composed of atomic and coupled models. DEVS models provide a basis for the design of event-based logic control. The event-based control paradigm is applied in advanced robotics and intelligent automation, showing how classical process control can be readily interfaced with rule-based symbolic reasoning systems (Zeigler 1989).

By illustrating a methodology in which the plant, its actuators, and sensors are described by discrete event models developed within the event-based control paradigm, a model of the controller is employed to validate its design in a plant/actuator/sensor experimental frame. The same model configuration is then employed for actual control operation by connecting the simulation executive, suitably modified, to a programmable controller that interfaces to the real plant/actuator/sensor system. This methodology is supported by the real-time interpretation of the DEVS formalism.

Several research applied DEVS to Real-Time systems, either by extension of the DEVS formalism (Hong et al. 1997, Sarjoughian and Gholami 2015) or by incorporating real-time functionalities in a DEVS environment (Yu and Wainer 2007, Niyonkuru and Wainer 2015). Special case tools are proposed in the literature for Real-Time DEVS (RT-DEVS) model analysis and design (Furfaro and Nigro 2008).

3 THE DEMES METHODOLOGY

Figure 1 shows the architecture of DEMES. A designer starts (1) by modeling the System of Interest (a RTS and its environment) using formal specifications (for instance DEVS, Bond Graphs, etc.). These models subsequently transformed into TA and verified using model-checking tools (2). In parallel with this formal verification phase, we use the same models to test the components in a simulated DEVS environment (3). The physical environment can also be simulated (4) together with the RTS model under particular loads (5). These tested submodels can be deployed incrementally into the target platform (6). Most of the testing phase (7) can be done using simulation (with faster than RT performance), even if the hardware is not available, if there are risks, or practical issues. Design changes are done incrementally in a spiral cycle (8), providing a consistent set of apparati throughout the development cycle. The cycle ends with the RTS fully tested, and every model deployed in the target platform (9).

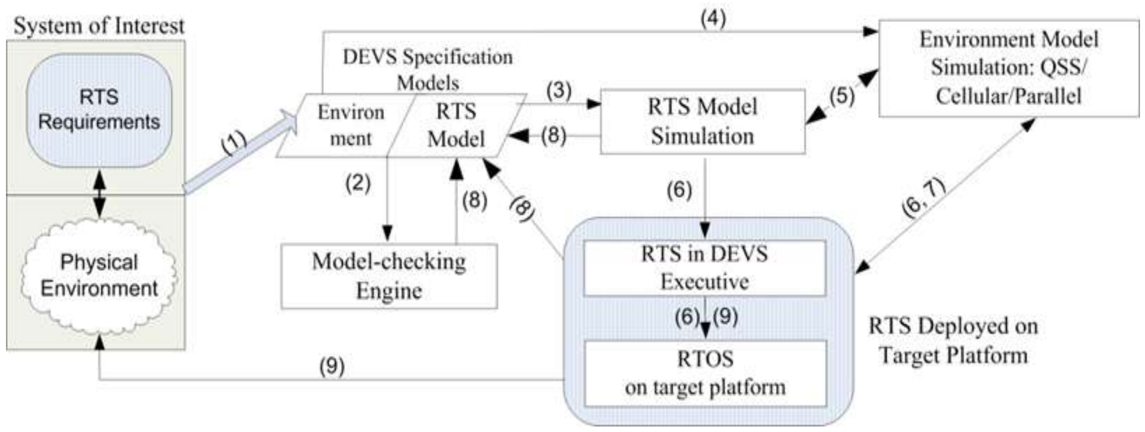

Figure 1. DEMES: Discrete-Event Modeling of Embedded Systems

The methodology rests on three pillars: model specification, analysis and execution based on DEVS theory. DEVS supports hierarchical and modular construction of models, which fits our needs (models at different levels of abstraction can be defined independently, and later integrated into a hierarchy). DEVS decouples model, experiments, and execution engines (allowing for portability and interoperability). The research used:

- RT Model analysis: we proposed the transformation of DEVS graphs into TA. This involved studying formal transformation of DEVS Graphs into semantically equivalent TA models, while maintaining their original structure and behavior, including continuous components. To use model checking, we needed to define a suitable finite abstraction of the hybrid system that can be verified and hence reachability can be computed. We proposed a new technique to represent the continuous system in discrete format using DEVS (Wainer and Saadawi 2010).

- RT Model execution: we built DEVS RT engines for the models above. In order to deploy components incrementally into varied hardware, we transform simulation models into executable models in the target, defining the mechanisms to map this runtime system into specialized hardware.

- We defined new methods for integrating models of complex physical systems and RTS, based on Cell-DEVS model specification and QSS and Hybrid model specifications.

- The models were integrated with parallel simulation engines. Although current experiments showed considerable speedups for simulating complex models, there is still open research on predictable performance to guarantee RT deadlines.

E-CD++ (Niyonkuru and Wainer 2015) implements DEMES, and it provides a platform for models to be defined according to the DEVS formalism and implemented in RT. For deploying the models on hardware, the tool allows for the generation of binary files that can be interfaced with input/output devices through the ARM MBED Library. Following, we show a case study, showing the methodology and results (which is mature enough so that undergraduate students are able to use it to develop applications).

3 CASE STUDY: A LIGHT RAIL CONTROLLER

In this section we show how to use the DEMES methodology described above for the modeling and implementation of a prototype light rail controller. Light rails, are a form of urban rail transit using rolling stock similar to a tramway but operates at a higher capacity with an exclusive right-of-way. A few light rail networks tend to have characteristics like rapid transit or even commuter rail; some of these rapid transit-like systems are referred to as light metros. Light rail systems are found throughout the world and have become popular in recent years because of their low capital cost and increased reliability when compared with heavy rail systems (Thompson 2003).

The key task of Railway is to transfer the passengers from one station to another. The stops of these stations are fixed and are controlled manually by a pilot. However, the issue with the current system is that it stops the Railway on every station even when there is no passenger boarding or disembarking from that station, which wastes a significant amount of travel time. In an attempt tackle this problem and make the Railway more efficient, we propose a stop control system in which the train would be stopped only when passengers need to board or get off from that stop.

We built a prototype based on a robot shield using the Nucleo F411RE board mounted on a seed shield robot (with an ARM Cortex M4 processor and 512kB of flash memory). A sensor controller activates or stops the light sensor, receives the sensor readings, and sends messages to the motor controller, specifying whether the robot is on track, off track, or has reached the destination. The controller also receives on/off track and stop signals from the sensor, and it sends appropriate commands to the motors. The Nucleo Board F411RE shown on the left side of Figure 2 has an ARM Cortex M4 processor and 512kB of flash memory with an STM32 microcontroller. The seed shield bot shown on the right side of the Figure has 5 Infra-Red reflectance sensors for line and edge following. In addition, the shield bot has two durable 160:1 micro metal gear motors and six Grove expansion ports for easy attachment of more sensors and actuators. It has 1 user LED shared with Arduino and 1 user and 1 reset push button. The power switch on the seed is used to turn on/off the robot. The charging port is used to charge the robot. Left and right motors enable the movement of the corresponding wheels. Grove ports are the expansion ports, Line finder Potentiometer is used to increase or decrease the sensitivity and five sensors for line and edge following.

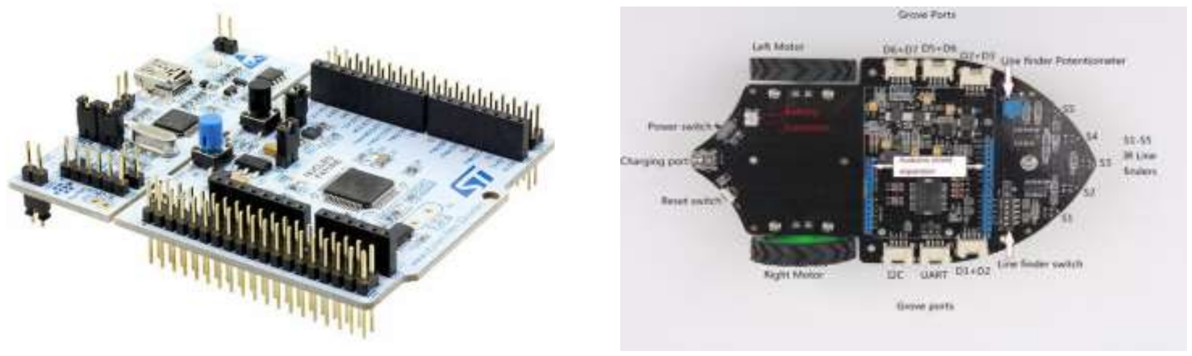

Figure 2. Nucleo Board F411RE and Seed Shield

Five Infrared sensors (S1, S2, S3, S4 and S5) were used as an input to the Railway. On receiving an input on any of the five sensors the train starts moving towards its requested destination. The sensors ports used are A0, A1, A2, A3, D4. The movement can also be controlled by digital push buttons. On pressing a push button, the train needs to start moving towards its requested station.

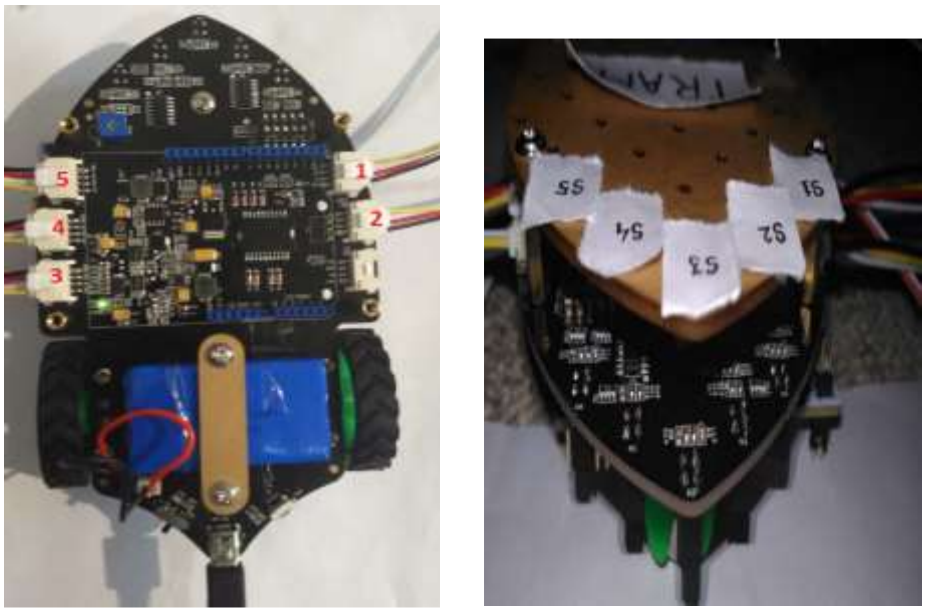

Figure 3. Digital Push Button Connections

3.1 DEVS MODEL DESCRIPTION

The model is made up of a stop controller and a wheel controller. The stop controller is responsible for deciding when to halt the train depending on the type of input signal received. The wheels controller is responsible for the Railway by deciding the direction of the movement whether it should move forward or backward, and it also decides what speed, and to compute the distance between the current and next station. The controller checks for passengers travelling inside the train and the outside passengers, who are waiting at the different stations to board the train.

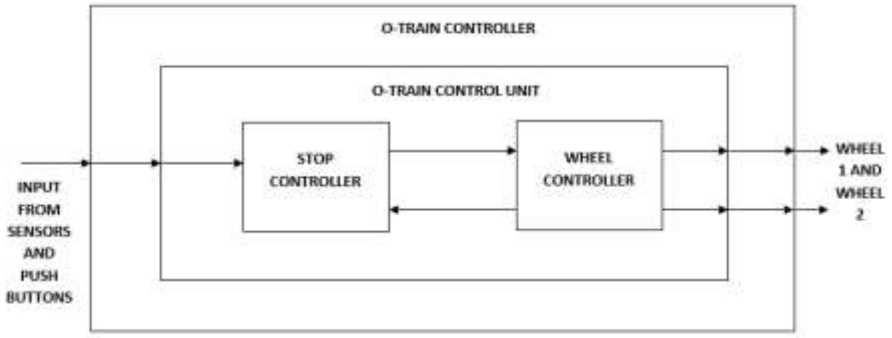

Figure 4. Top Model Structure

Figure 4 shows the top model, and two atomic components: Stop Controller and Wheel Controller. The Stop Controller model gives the output to the Wheel Controller model. Also, it receives a feedback from the Wheel controller model. The Wheel Controller model sends the output to the wheels according to the inputs received from the Stop Controller. Stop Controller model activates the sensors on receiving the start input on one of its ports. The atomic model receives the inputs from the sensors/ push buttons and gives the output to the atomic model Wheel controller. The stop controller atomic model is shown in Figure 4.

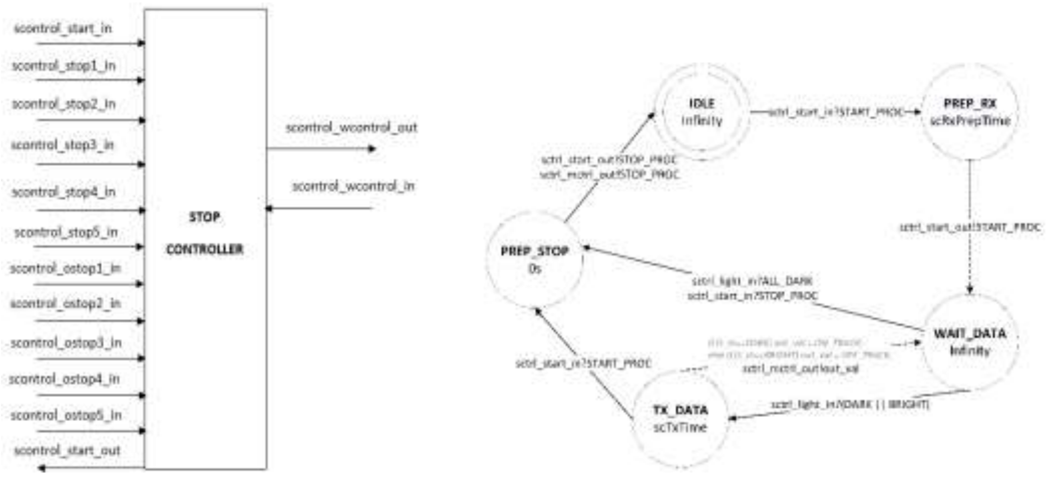

Figure 5. Stop Controller Atomic Model and DEVS graph

Figure 5 shows the structure of the Stop Controller Atomic model, and a DEVS Graph representing the sensor controller’s behavior. As discussed earlier, the DEVS graph can be transformed into a timed automaton and checked using TA model checking tools like UPPAAL. Then, we can use the same model to run a simulation using the CD++ software (Wainer 2009) and extensions. We illustrate the execution mechanism using trace logs collected during the execution of the railway in Figure 6. Once tested in simulation mode, the same models are deployed onto the device.

```
DRIVER: INPUT MESSAGE  Time: 00:02:10:403:002
 Port: start_in Value: 11
 - advance_execution()::STController
 - advance_execution()::IR_Sensor
 model->external() model->advance(): 00:000:000
 - collect_outputs()::STController
 - advance_execution()::STController
 - collect_outputs()::IR_Sensor      model->out()
 - advance_execution()::IR_Sensor    model->internal() model->advance(): ...
 - advance_execution()::mctrl
 model->external() model->advance(): 00:000:000
 - collect_outputs()::STController
 - collect_outputs()::mctrl  model->out()
DRIVER: OUTPUT MESSAGE  Time: 00:02:10:403:559
 Port: motor Value: 0
```

Figure 6. Simulated results

We declare the pins used by the hardware, and link it using the MBED library to define the input controller models (Stop Controller, and Wheel Controller). 5 pins associated with the IR sensors are initialized as stop sensors, and the pins associated with the different grove ports are initialized as the push buttons’ inputs. We define the pins linked with the two motors to give these pins the output of the controller, also defined the pins to enable these motors first. After the initialization stage is completed, we need to define ports and drivers, in this implementation, we define port drivers and output drivers and instantiate the driver for a polling period. After the drivers have been defined and instantiated, the DEVS models are implemented, we implement the internal and external transition functions. Then the port definitions that connect the physical port drivers to the DEVS model are implemented after all the required ports have been instantiated. The top model, which defines the links between the control unit coupled model and the systems I/O port driver is created.

A number of videos showing the result on the target platform are at http://www.youtube.com/arslab, as in Figure 7.

The Train rests at station 1 until and unless an input through sensors or push buttons is given to it. When an input is provided to one of the infra-red sensors (A0, A1, A2, A3, D4), the train starts moving forward. Each sensor represents a station; therefore, the next destination is determined by input to a sensor. Similarly, there are five push buttons, each of which represents a station. When a button is clicked the train starts moving forward or backward according to the requested destination but would only change direction when it has reached the last station or the first station depending on the direction requested. The Train can be controlled by using a sensor or a button for both forward and backward movements. It can also be controlled by using sensor for forward movement and button for backward movement or vice versa but both sensor and button cannot be used simultaneously. The complete real time model structure with the train, its path, different stations along with the push buttons of respective stops.

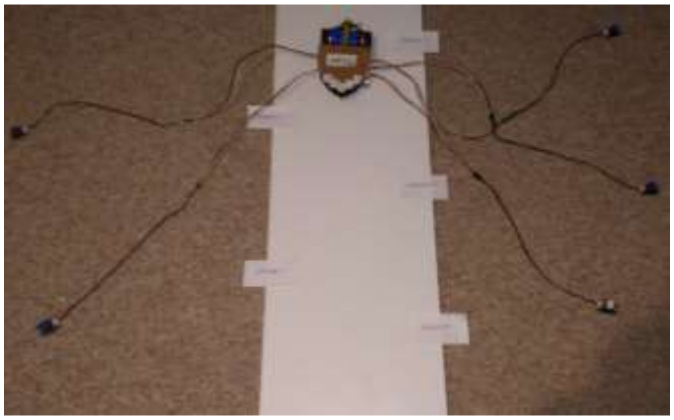

Figure 7. Train Movement, different stations and their respective push buttons

When the Train is at rest a request arrives from the inside passenger. An input is received by the sensor and the train moves to the requested station (station 2). Next requested station is station 3 by the inside passenger. On receiving a sensor input the train further moves towards the station 3. The subsequently requested stations are station 4 and 5 by the inside passengers. Therefore, the train travels to the station 4 from station 3 and then to the station 5 from station 4. Now the inside passenger requests for station 3, therefore, the train moves backward towards the station 3 from station 5 without stopping at Station 4. Next request arrives for station 4 through the push button but the train does not move forward towards the station 4 because it is going in the opposite direction and not at station 1 yet. The inside passengers now request for station 1, therefore, the train moves from station 3 to station 1. An input is received by the sensors for station 3 again. The train moves forward from station 1 to station 3 and then fulfils the initial station 4 requests.

7 CONCLUSION

We have introduced the DEMES methodology, and showed an implementation of a Railway control system using DEMES. We showed that DEVS is useful in simplifying the development of complex systems like Railway controller. The models are reusable and we were able to adapt a previously developed robot cart controller.

REFERENCES

Furfaro, A.; Nigro, L. 2008. Embedded Control Systems Design based on RT-DEVS and temporal analysis using UPPAAL, 2008 International Multiconference on Computer Science and Information Technology. Wisla, Poland.

Hong, J., Song, H., Kim, T., and Park, K. 1997. A real-time discrete event system specification formalism for seamless real-time software development. Discrete Event Dynamic Systems, 7(4), pp.355-375.

Niyonkuru, D. and G. Wainer. 2015. Discrete-Event Modelling and Simulation for Embedded Systems. Computing in Science & Engineering, 17(5), pp.52-63.

Niyonkuru, D. and G. Wainer. 2015. "Towards a DEVS-Based Operating System". SIGSIM PADS '15, 101–112. London, UK.

Niyonkuru, D., & Wainer, G. 2016. A kernel for embedded systems development and simulation using the boost library. In Proceedings of the Symposium on Theory of Modelling & Simulation. Washington, DC.

Saadawi, H.; Wainer, G. 2009. "Verification of Real-Time DEVS Models". SpringSim'09, San Diego, CA.

Saadawi, H.; Wainer, G. 2010. "Rational Time-Advance DEVS (RTA-DEVS)". Symposium on Theory of Modeling and Simulation, Orlando, FL.

Sarjoughian, H., & Gholami, S. 2015. Action-level real-time DEVS modeling and simulation. Simulation, 91 (10), pp. 869-887.

Thompson, G. 2003. Defining an Alternative Future: The Birth of the Light Rail Movement in North America. Transportation Research Circular. Transportation Research Board (E–C058).

Yu, Y., Wainer, G. 2007. "eCD++: an engine for executing DEVS models in embedded platforms". In Proceedings of the 2007 Summer Computer Simulation Conference. San Diego, CA.

Wainer, G. 2009. Discrete-event modeling and simulation: a practitioner's approach. CRC press.

Wainer, G. 2015. DEVS modelling and simulation for development of embedded systems. In Proceedings of the 2015 Winter Simulation Conference. Huntington Beach, CA.

Zeigler, B., Kim, T.G., & Prähofer, H. 2000. Theory of modeling and simulation. Academic Press.

Zeigler, B. 1989. DEVS Representation of Dynamical Systems: Event-Based Intelligent Control. Proceedings of the IEEE (Volume: 77, Issue: 1).

ACKNOWLEDGEMENTS

This research has been partially funded by NSERC. We want to thank N. Kaushal and V. Joshi who participated in the development of the railway model and controller.

AUTHOR BIOGRAPHIES

GABRIEL A. WAINER, FSCS, SMIEEE, received the M.Sc. 1993) at the University of Buenos Aires, Argentina, and the Ph.D. 1998, with highest honors) at UBA/Université d'Aix-Marseille III, France. In July 2000, he joined the Department of Systems and Computer Engineering at Carleton University (Ottawa, ON, Canada), where he is now Full Professor and Associate Chair for Graduate Studies. He has held visiting positions at the University of Arizona; LSIS (CNRS), Université Paul Cézanne, University of Nice, INRIA Sophia-Antipolis, Université de Bordeaux (France); UCM, UPC (Spain), University of Buenos Aires, National University of Rosario (Argentina) and others. He has published around 400 research articles and five books in the field of Modeling and Simulation. He is one of the founders of the Symposium on Theory of Modeling and Simulation, SIMUTools and SimAUD. Prof. Wainer was Vice-President Conferences and Vice-President Publications, and is a member of the Board of Directors of the SCS. Prof. Wainer is the Special Issues Editor of SIMULATION, member of the Editorial Board of IEEE Computing in Science and Engineering, Wireless Networks (Elsevier), Journal of Defense Modeling and Simulation (SCS. He is the head of the Advanced Real-Time Simulation lab, located at Carleton University's Centre for advanced Simulation and Visualization (V-Sim. He has been the recipient of various awards, including the IBM Eclipse Innovation Award, SCS Leadership Award, and various Best Paper awards. He has been awarded Carleton University's Research Achievement Award 2005, 2014), the First Bernard P. Zeigler DEVS Modeling and Simulation Award, the SCS Outstanding Professional Award 2011), Carleton University's Mentorship Award 2013), the SCS Distinguished Professional Award 2013), and the SCS Distinguished Service Award 2015. He is a Fellow of SCS.

JOSEPH BOI-UKEME is a Ph.D. student in the Department of Systems and Computer Engineering, Carleton University. He has a B.Eng. in Electrical and Electronics Engineering and a M. Eng. In Electronics and Telecommunications Engineering from the University of Benin, Nigeria. He holds an MSc in Petroleum Engineering from IFP, France. Previously he was a lecturer at the University of Benin, Nigeria, and a Field Engineer for Schlumberger.