Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

DEVS MODEL DESIGN FOR SIMULATION WEB APP DEPLOYMENT

Laurent Capocchi
Jean-François Santucci
Johanna Fericean

University of Corsica
UMR CNRS 6134
Corte, 20250, FRANCE

Bernard P. Zeigler

RTSync Corp.
Potomac, MD 20854, USA

ABSTRACT

The discrete-event system specification (DEVS) formalism has been recognized to be able to enable a formal and complete description of the components and subsystems of hybrid models. What is missing for accelerated adoption of DEVS-based methodology is to offer a way to design web apps to interact with a simulation model and to automatically deploy it on an online server which is remotely accessible from web app. The deployment of DEVS simulation models is the process of making models available in production where web applications, enterprise software, and APIs can consume the simulation by providing new inputs and generating outputs. This paper proposes a framework allowing one to simplify the DEVS simulation model building and deployment on the web by the modeling and simulation engineers with minimal web development knowledge. A case study on the management of COVID-19 epidemic surveillance is presented.

1 INTRODUCTION

Modeling and simulation (M&S) of complex systems is a discipline dedicated to the field of engineering and research that tends to be increasingly exploited by modelers, end users, and developers of web interface applications (or mobile apps) (Wang and Wainer 2016; St-Aubin et al. 2019). Initially, models and simulators were dependent on a specific application domain, and they were developed by a team of engineers/researchers specialized in a given field of application. Therefore, the analysis and development of a simulation model and its simulator were carried out by specialists whose working environment was confined to the workstation of a research laboratory or company. This approach does not allow one to provide simulation models that can be used to a large scale of non-specialist end users. These simulation models must be put into production so that end users can access them more easily. Deploying them so they are accessible to end users is an incredibly important skill for engineers/researchers to have. While simulation models building is a task which is well defined and mastered by the M&S engineers, a web deployment of them requires specific capabilities involving at least basic understanding of full-stack development and integration in order to achieve simulation model deployment. Basically, a simulation model web deployment implies to implement a web app (allowing I/O specification, simulation results visualization and analysis in real time) and a simulation execution process based on a bidirectional channel in order to allow the simulation model to communicate with the web app. These two implementations (bidirectional communication and web app) are usually difficult to accomplish by M&S engineers which, in any case, that are not under his responsibility.

This article proposes a methodology that enables simplifying the building and deployment of simulation models on the web by the M&S engineers with minimal knowledge of web development / deployment. The proposed approach is based on three steps: (i) creating the web apps from the simulation model specification using a design block approach in a semi-automatic way; (ii) building a standalone simulation model using a generic framework allowing the definition of design patterns that automatically perform the event-based bidirectional communication between the simulation process and the end user through a web app; and (iii) deploying this simulation model and the corresponding web app on a cloud platform.

The proposed approach requires a generic M&S environment which allows modular and hierarchical specifications of complex systems to be easily defined both as a building block and as an event-driven modeling process. The DEVS formalism (Zeigler et al. 2018) has been recognized to be capable of a formal and complete description of the components and subsystems of hybrid models. What is missing for accelerated adoption of DEVS-based methodology for complex system design is to offer a way to automatically deploy DEVS models on an on-line server which is remotely accessible/executable from web apps (progressive mobile application or responsive web application) for simulation and an automatically design of associated web interfaces. The originality of the approach leans on the development of building blocks and architectural patterns based on a library of reusable DEVS components. This methodology, recently introduced by Zeigler in (Zeigler 2021), "offers a notional architecture for intelligent hybrid complex system design and proceeds to focus on the decision layer to consider DEVS models for basic behaviors such as choice of alternatives, perception of temporal event relations, and recognition and generation of finite state languages cast into DEVS time segments". The methodology presented in this paper is therefore based on the DEVS formalism mainly due to its building block and architectural pattern capabilities. It allows us to define: (i) a specific design pattern used to implement bidirectional channel and (ii) a building block functionality used to build web apps involved in the simulation model web deployment.

The paper is organized as follows. Section 2 is dedicated to related work, while Section 3 gives an overview of the proposed approach. Section 4 introduces the basic concepts developed by presenting in detail both: (i) the creation of the web simulation model based on a specific DEVS design pattern and a way to implement bidirectional communication and (ii) the deployment of the web simulation model based on the definition of a DEVS Web Builder. A case study on the surveillance of the COVID-19 epidemic is given in Section 5. Finally, Section 6 concludes and provides some perspectives of the work.

2 RELATED WORK

In the domain of discrete-event M&S a set of work based on a client-side web application approach allows defining user interfaces to access, visualize, and analyze the results of models executed on a server and also to perform model simulations within the browser itself. The use of web applications for the purpose of simulation modeling is well described in (Byrne et al. 2010). The paper illustrated client-side approaches to improve user interfaces and to perform simulations and visualizations within a browser. They describe the use of client-side technologies to create an interactive web application for a specific (not generic) simulation model of biochemical oxygen demand and dissolved oxygen in rivers.

In (Shaikh et al. 2021) the authors present a single web application for executing a broad range of models, including the possibilities of having access to simulation tools through web apps. The paper also provides features for sharing simulations and interactively visualizing their results. They plan for the future to use this tool to foster reproducibility, stimulate collaboration, and facilitate the creation of simulation models. Another example of web application tool simulation is presented in (Ha et al. 2019). The paper describes a web-based application developed to allow plant biologists to construct dynamic mathematical models of molecular networks, interrogate them in a manner similar to what is done in the laboratory, and use them as a tool for biological hypothesis generation through the proposed tool. It enables users to build, validate, and use intuitive qualitative dynamic computer models through a graphical user interface that does not require mathematical modeling expertise. It is research work to consider, but does not propose a generic tool and is only dedicated to biological M&S. In (Uran and Jezernik 2006), authors propose web apps designed and compiled into the MATLAB software suite in order to propose web applications based on browsers to end users who are not MATLAB developers. MATLAB Compiler allows one to share MATLAB Web apps with end users who do not have MATLAB software. They propose a first tool called the MATLAB App Designer which is used to implement the web app interface. The code is then compiled with a MATLAB compiler to deploy the complied package in a MATLAB Web App Server. The web app is available through an url and can be shared via a classic browser. This approach requires you to install the web app server previously. Visualizing results is quite difficult, since displaying graphics in MATLAB App Designer requires a different workflow, which can be included only in the MATLAB command line.

The DEVS formalism (Zeigler et al. 2018) was introduced by Zeigler in the 1970s to model discrete event systems hierarchically and modularly. DEVS formalizes what a model is, what it must contain, and what it does not contain (experimentation and simulation control parameters are not contained in the model). Moreover, DEVS is universal and unique for discrete-event system models. Any system that accepts events as input over time and generates events as outputs over time is equivalent to a DEVS model. Currently many DEVS M&S engines offer a programming interface (API) that is user-friendly to define new models using a high-level language (Risco-Martín et al. 2017). However concerning the web M&S in the framework of the DEVS area, few environments provide a user-friendly graphic user interface (GUI) that integrates web capabilities In (Seo and Zeigler 2009), the authors propose an automatic generated DEVS interface to web services to apply a formal method to orchestrating web services to provide a dynamic testing environment of business processes with DEVS M&S and to enhance the reusability of business processes and the interoperability of instances of BPEL (Business Process Execution) language. The paper introduces a way to consider integration of business processes as DEVS M&S through mapping web services to DEVS atomic models. In (Seo et al. 2015), the same authors present the basic ideas of Web-based Simulation Systems consisting of databases for model information and a web simulator on a web server. Also has to be mentioned the work presented in (Sarjoughian and Zeigler 1998) that deals with a web-enabled DEVS environment (called DEVSJAVA) based on the DEVS formalism and implemented in Java language. DEVSJAVA offers the ability to design simulation models based on web technology. The tool provides a graphically oriented interface through which users implement discrete-event models. It facilitates model development and simulation independently of the hardware platform using Java-enabled browsers. Furthermore, as in (Seo et al. 2015), it provides the foundation to enable collaborative M&S. The paper (Capocchi and Santucci 2021) proposes an architecture to perform discrete event simulations using mobile devices. The connection between M&S environment, Cloud and smartphones is performed via a web services oriented architecture. The discrete-event models are remotely accessible from smartphones via web services. A smartphone is part of the simulation model and can be used for real data acquisition or to control the model during the simulation. In this work, no solution is proposed to help the end user define a web application that interfaces with the simulation model.

The paper (Al-Zoubi and Wainer 2015) provides a RESTful API for integrated simulation environments, particularly DEVS-based tools. It describes how the authors used two components: (i) the RESTful Web Services framework API to be able to communicate through the Web using REST style and (ii) the access to the DEVS tool via the REST API to run simulation experiments. To enhance simulation interoperability on the Web, the authors implemented the RISE (RESTful Interoperability Simulation Environment) middleware.

The previously mentioned work brings a lot of information and interesting ways to connect M&S with web apps. Therefore, we have been inspired by the paper (Seo et al. 2015). However, a set of issues should still have to be solved: (i) the transfer of input data from the end user or some data storage or connected object into the model to be simulated from a web application, (ii) the design of a web app associated with a given model to be simulated, and (iii) the display of the simulation results on-line in a web app. The contribution of the paper meets the problem associated with the three points mentioned above. It will result in a generic tool for Web M&S in a similar way as proposed in (Singh et al. 2019) on the aspects of Web App Builder. Even if the paper does not deal with simulation, the authors propose the development of a web GIS interface using Web AppBuilder.

3 OVERVIEW OF THE APPROACH

Figure 1 gives an overview of the proposed approach, which involves three ordered steps introduced to help the engineer when deploying the DEVS simulation model in a cloud: (i) building a given DEVS model allowing a bidirectional channel between a web user app and a standalone DEVS simulation model; (ii) designing an user interface which will be associated with the standalone DEVS simulation model; and (iii) deploying the solution on the Cloud (using a dockerization process, for example).

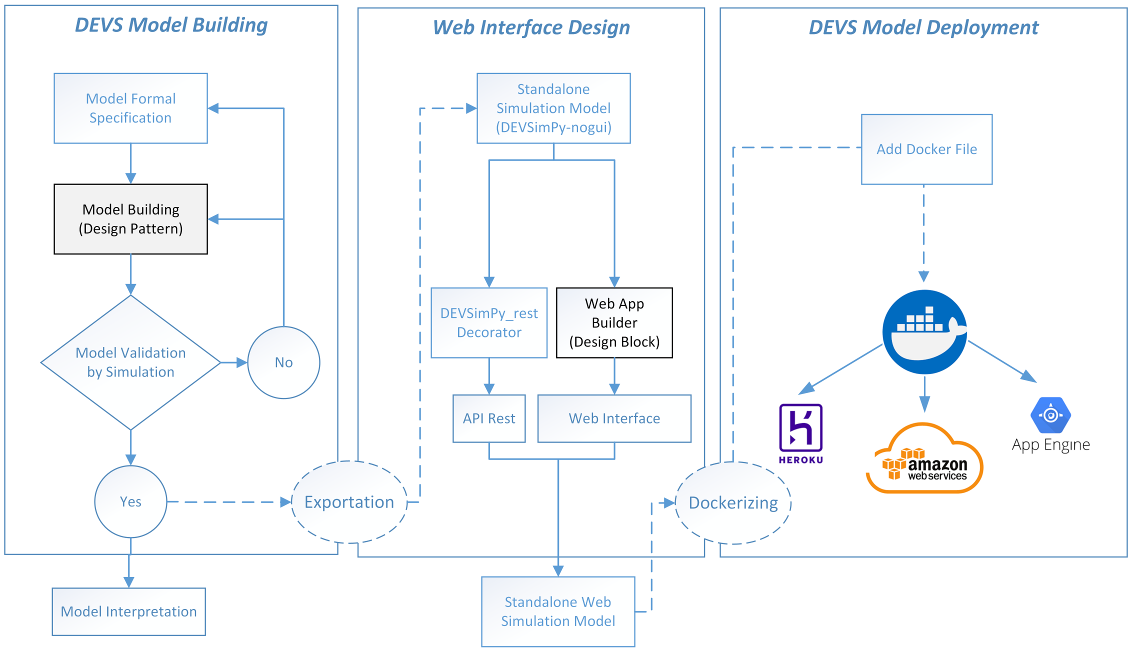


Figure 1: Proposed approach: DEVS Model Building & Design based on Design Pattern approach and DEVS Model deployment to the Cloud using Web App Builder and dockerizing process.

This paper deals with the first and second steps since the third step is straight-full once the two first ones have been realized. It describes an approach for facilitating DEVS model simulation on the Web through a web app, which can be hosted in a cloud solution through dockerization. The proposed approach does not focus on the last technical point, as several cloud solutions are easily compatible with the presented implementation.

The definition of the building block and the design pattern is facilitated by the DEVS capability in terms of the composition of the model. With DEVS, a model of a large system can be decomposed into smaller component models with couplings between them. DEVS formalism defines two kinds of model: (i) atomic models that represent the basic models providing specifications for the dynamics of a subsystem using state transition functions, and (ii) coupled models that describe how to couple several component models (which can be atomic or coupled models) together to form a new model. As presented in the left part of the figure 1, the DEVS Model Building starts from the DEVS formal specifications of the complex system under study that are used by the Model Building block in charge to decorate the block model to make it accessible on the Web. We defined a Design Pattern involving basic DEVS-based blocks inspired by (Zeigler 2021) in order to design a web-oriented simulation model (which has bidirectional communication through the web). If the resulting simulation model is valid, it can be transformed into a standalone version (*Exportation* in Figure 1). The DEVS Model Building process can be performed with the DEVSimPy environment. DEVSimPy (Python Simulator for DEVS) (Capocchi et al. 2011; Capocchi L.) is an user-friendly interface for collaborative M&S of DEVS systems implemented in Python language (Nagpal and Gabrani 2019). DEVSimPy has been set up to facilitate both the coupling and reusability of the DEVS models. With DEVSimPy, DEVS models can be stored in a library in order to be reused, shared, and defined design patterns. Moreover, DEVSimPy implements a functionality to automatically export their simulation models in a standalone containerized package. Once the model has been validated using simulation, the user can switch to the second step consisting in the design of an associated web app (”Web Interface Design” block in the middle part of the figure 1).

In the Web Interface Design process, the standalone simulation model can be executed with the DEVSimPy-nogui (Capocchi L.) software which is a non-graphical, command-line execution version of DEVSimPy. More precisely, the standalone simulation model is a compressed file that embeds DEVSimPy-nogui files, the yaml version of the simulation model, and the DEVSimPy model libraries whose simulation model depends. These libraries can be associated with a simulation model and must be included in the standalone version. DEVSimPy has a set of domain-specific model libraries which have already been developed and which can be used to implement a simulation model. The standalone version can be used to generate an API Rest (Representational State Transfer) (Belkhir et al. 2019; Segura et al. 2018) service (DEVSimPy_rest) as described in (Capocchi and Santucci 2021) to remotely run the DEVSimPy simulation model. Furthermore, it can be also used to semi-automatically generate web apps through the Web App Builder by parsing its yaml file. This paper deals with this last aspect, while the Rest API has already been presented in (Capocchi and Santucci 2021). Semi-automatically means that a first version of the web app is obtained by parsing the yaml file, but it can be modified/extended by the developer using the Web App Builder web interface (described in Section 4) in order to produce a web app to interact with the standalone simulation model called ”standalone web simulation model”. The standalone Web simulation model can be used to build a Docker container that can be deployed in a cloud (Heroku (Middleton and Schneeman 2013), AWS (Bankar 2018), Google App Engine (Gupta et al. 2020), etc.) to be accessible to end users. The next section describes in detail both the DEVS Model Building and the Web Interface Design processes.

4 DEVS MODEL BUILDING METHODOLOGY FOR WEB DEPLOYMENT

This section describes the basic concepts involved in the DEVS Model Building Methodology for the web deployment of standalone DEVS simulation models. First, the Design Pattern approach allowing to automatically generate the standalone DEVS simulation model is described. Then the generation of such a web app is explained.

4.1 DEVS Design Pattern based Model Building

In order to help the simulation team deploy a DEVS simulation model on the web, we propose a design pattern (DEVS-TO-WEB Decorator) made up of two types of DEVS atomic models: *Web Generator* and *Web Collector*. The DEVS Web Generator atomic model is responsible for retrieving data from the end user web application (1 in figure 2) and the DEVS Web Collector atomic model is dedicated to collect simulation results and send them back to the end user web app (2 in figure 2). The Web Generator embeds a loop process which waits for an event coming from the end user web app. It transmits this event to the DEVS Coupled Model connected to the Web Collector, which will send back the simulation results to the Web app.

In the ”DEVSimPy M&S” top layer of figure 2, the engineer designs a DEVS coupled model of the complex system under study by using classic DEVS Generator and Collector models to validate by simulation the DEVS coupled model corresponding to the complex system. Then, in the ”DEVS-TO-WEB Design” layer, the DEVS-TO-WEB Decorator can be applied to replace these classic DEVS models by the new Web Generator and Web Collector models. Moreover, the engineer needs to make some settings related to the network configuration in these two models in order to enable the directional communication between the simulation process and the web app (mainly URL network settings as explained in the next section).

Figure 3 shows how the bidirectional communication mechanism is implemented using the socket client/server message over the HTTP protocol and the Python libraries. When an end user wants to send a message to a standalone simulation model stored on an asynchronous web server (1 in Figure 3), the web

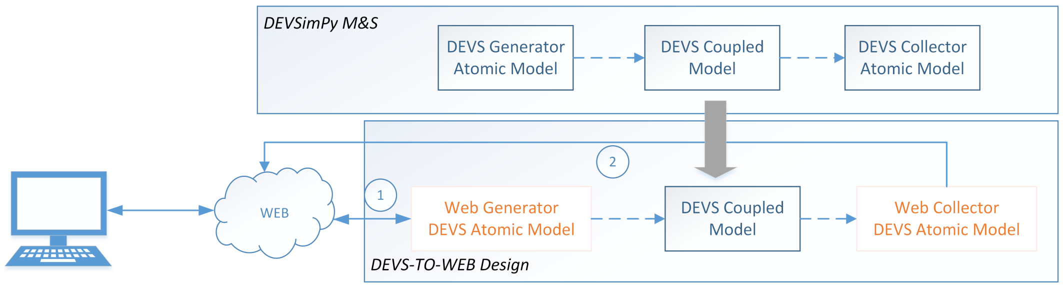

DEVSimPy M&S
DEVS Generator Atomic Model
DEVS Coupled Model
DEVS Collector Atomic Model
WEB
1
2
Web Generator DEVS Atomic Model
DEVS Coupled Model
Web Collector DEVS Atomic Model
DEVS-TO-WEB Design

Figure 2: DEVS Model Building Methodology using DEVS-TO-WEB Decorator involving the DEVS *Web Generator* and *Web Collector* models introduced to enable the bidirectional communication in real time.

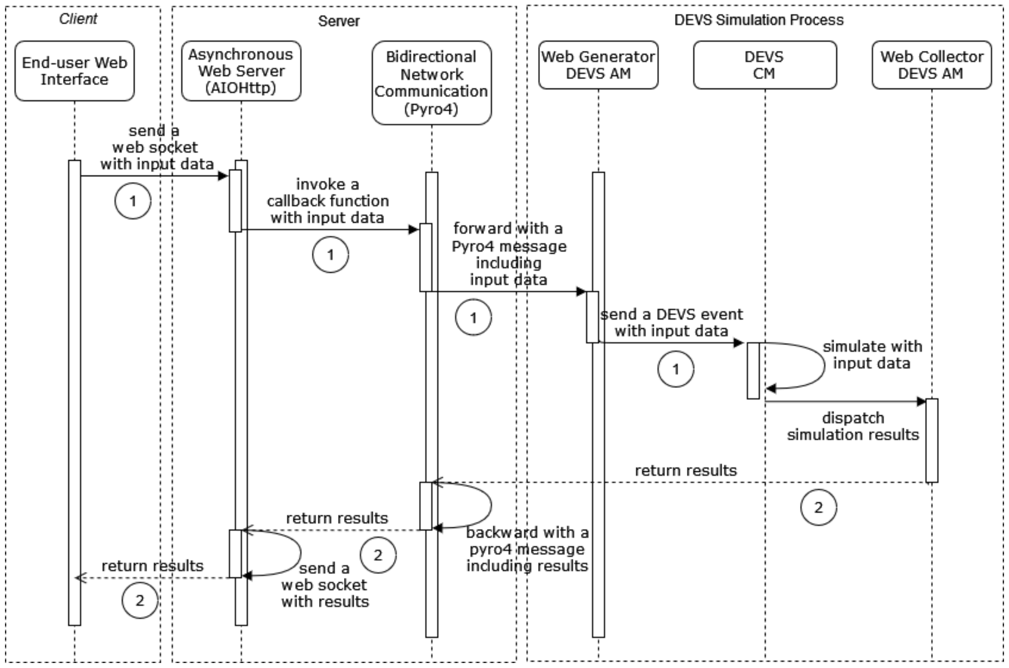

Client
Server
DEVS Simulation Process
End-user Web Interface
Asynchronous Web Server (AIOHttp)
Bidirectional Network Communication (Pyro4)
Web Generator DEVS AM
DEVS CM
Web Collector DEVS AM
send a web socket with input data
invoke a callback function with input data
forward with a Pyro4 message including input data
send a DEVS event with input data
simulate with input data
dispatch simulation results
return results
backward with a pyro4 message including results
return results
send a web socket with results
return results
1
2

Figure 3: UML sequence diagram of the bidirectional communication between the end user web app and the simulation process based on the *Web Generator* and *Web Collector* DEVS atomic models. (1) Message passing of the input data from the end user web app to the DEVS Coupled Model (CM) and (2) the backward of the simulation results from the *Web Collector* to the end user web app.

application sends a socket to the AIOHttp (AIOHttp developers) web server that transforms this request into a Pyro4 (PyRo4 Developers) request sent to the Web Generator server embedded in a standalone simulation process. The AIOHttp Python package is asynchronous and designed to make the most out of non-blocking network operations. The Pyro4 Python package allows us to build applications in which objects can talk to each other over the network. The simulation extracts the data from the received message and the coupled model is executed to generate an output message to the Web Collector model. This model sends a message to a Pyro4 server (executed on the AIOHttp web server) that sends a socket to the web app to update it dynamically (2 in figure 3). The next subsection presents the basic concepts involved in automatizing the design of a web application associated with a given DEVS simulation model.

4.2 DEVS-Driven Web App Builder

A Web App Builder algorithm is used to generate a web app using Design HTML Blocks (HTML tags) organized in a smart way by parsing a yaml version of the simulation model exported from the DEVSimPy environment. First, the engineer creates an initial web app where he has to: (i) give a title (of the simulation, for example) and (ii) give the yaml file corresponding to the DEVSimPy simulation model. Then the Web App Builder algorithm generates a first version of the web app according to the content of the yaml file. For example, the Web App Builder algorithm transforms each web Generator DEVS model input message with a numeric type into input HTML tags (<input type="number">).

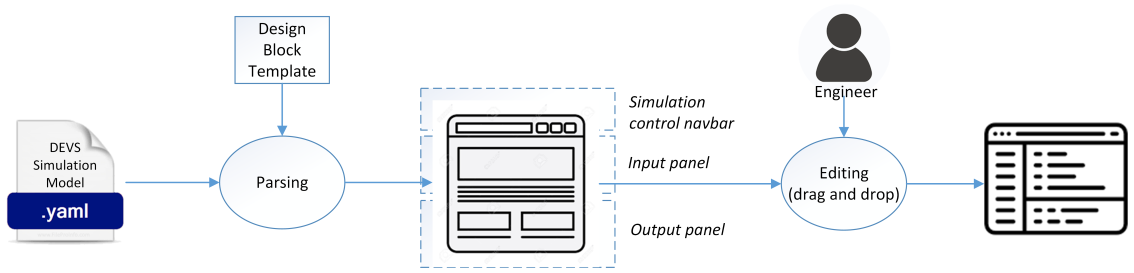


Figure 4: Web Interface Building from the DEVS simulation model (yaml file) to the end user web app.

Figure 4 gives an overview of the process that enables us to help design an end-user web app. A first step consists of a parsing process resulting in a first draft of an end-user web app. The parsing process takes as input both the yaml file corresponding to the DEVS simulation model created in the previous step and a set of Design Block templates defined in order to guide the user to build a web app associated with a given DEVS simulation model.

Three types of template are proposed: (i) the simulation control navigation bar composed of URL internal links to navigate inside the web interface; (ii) the input panel with all of the input fields used to get all values embedded in the message dedicated to the Web Generator DEVS model; and (iii) the output panel with all output fields used to display the value from Web Collector DEVS model inputs (simulation results). The designer can then modify the proposed web app using drag and drop of elements offered to him through a Web App Builder GUI.

5 CASE STUDY: EPIDEMIC COVID-19 SUPERVISION

The case study concerns Health System M&S and more specifically the notion of pathways. Pathways is a concept that can be used to coordinate the activities of otherwise uncoordinated agencies and services by focusing on the progress and outcomes of individual clients as they traverse such care organizations (Elbattah et al. 2018; Zeigler 2018).

Pathways are unique in that the results are tracked at the level of the individual being served. Each step of the Pathway addresses a clearly defined action towards problem resolution. Pathways have been developed for many issues, including homelessness, pregnancy, medical home, immunizations, lead exposure, childhood behavior issues, just to name a few. One client (or patient) may be assigned to many different Pathways depending on the problems identified. At first glance, Pathways may resemble clinical guidelines or protocols. The case study deals with the pathways involved in the surveillance of epidemic COVID-19. It involves 50 patients who are supervised using a pathway template based on six questions. We will only describe the parts of the case study development cycle that deal with aspects of the web application.

5.1 DEVS Model Building

As depicted in figure 5, the Pathway simulation model is composed of the *PathWays* DEVS coupled model (detailed in figure 7) and the three following DEVS atomic models: *Web Generator*, *Web Collector* and *Pathway_Filter*. Each *Patient* DEVS atomic model included in the *PathWays* DEVS coupled model implements a simple pathway process based on questions / responses. The *Web Generator* (resp. *Web Collector*) models, detailed previously, are dedicated to collect messages (resp. to inject simulation results) from (resp. to) the web app. The *Patient_Filter* DEVS atomic model redirects the input messages to the correct patient atomic model instantiated in the *PathWays* coupled model.

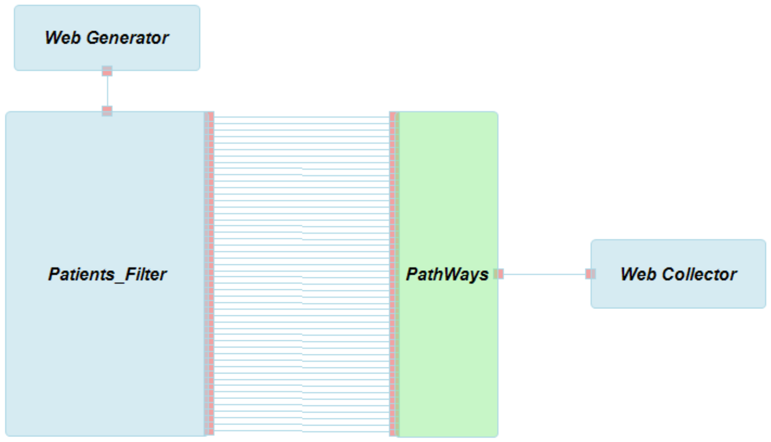


Figure 5: DEVSimPy simulation model with the *Web Generator* and *Web Collector* DEVS atomic models used in order to implement the web app/simulation bidirectional communication channel. The *Pathways* coupled model is described in Figure 7.

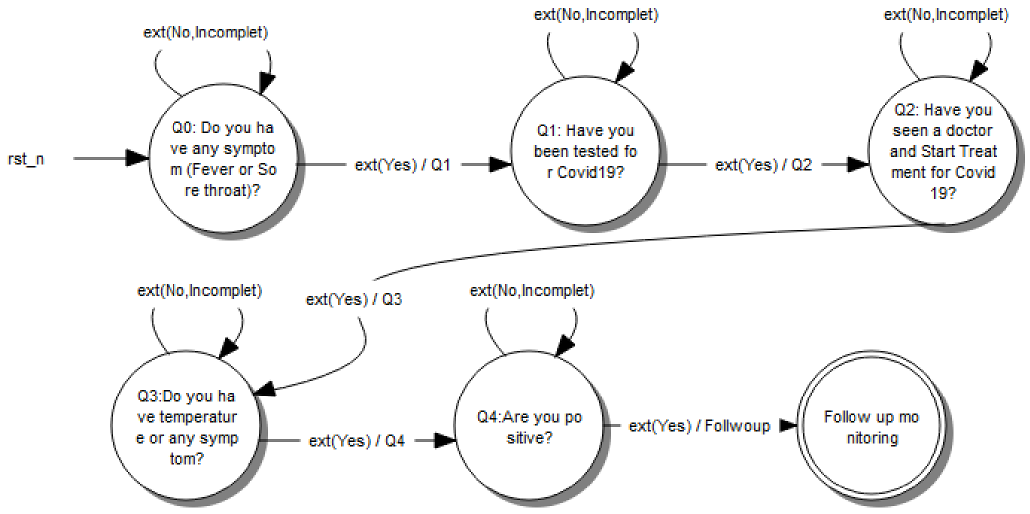

rst_n
ext(No,Incomplet)
Q0: Do you have any symptom (Fever or Sore throat)?
ext(Yes) / Q1
Q1: Have you been tested for Covid19?
ext(Yes) / Q2
Q2: Have you seen a doctor and Start Treatment for Covid 19?
ext(Yes) / Q3
Q3:Do you have temperature or any symptom?
ext(Yes) / Q4
Q4:Are you positive?
ext(Yes) / Follwoup
Follow up monitoring

Figure 6: Finite state automaton of a patient pathway (associated with a DEVS patient atomic model) with 6 states (number of questions).

A patient is a DEVS atomic model (Patient_i $i \in [1,50]$ in figure 7) dedicated to the modeling of the state automaton associated with 6 questions as described in figure 6: Question 1: ”Do you have any symptom Fever or Sore throat”; Question 2: ”Have you been tested for COVID-19”; Question 3: ”Have you seen a doctor and started treatment for COVID-19”; Question 4: ”Do you have temperature or any symptom”; Question 5: ”Are you negative”; Question 6: ”Follow up monitoring“.

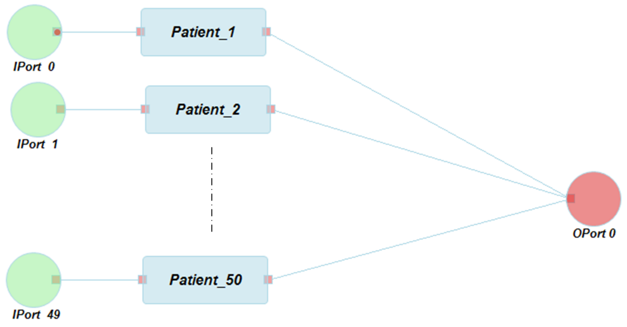


Figure 7: PathWays DEVS coupled model with the 50 patient DEVS atomic models whose the finite state automation associated with each atomic model is presented in Figure 6.

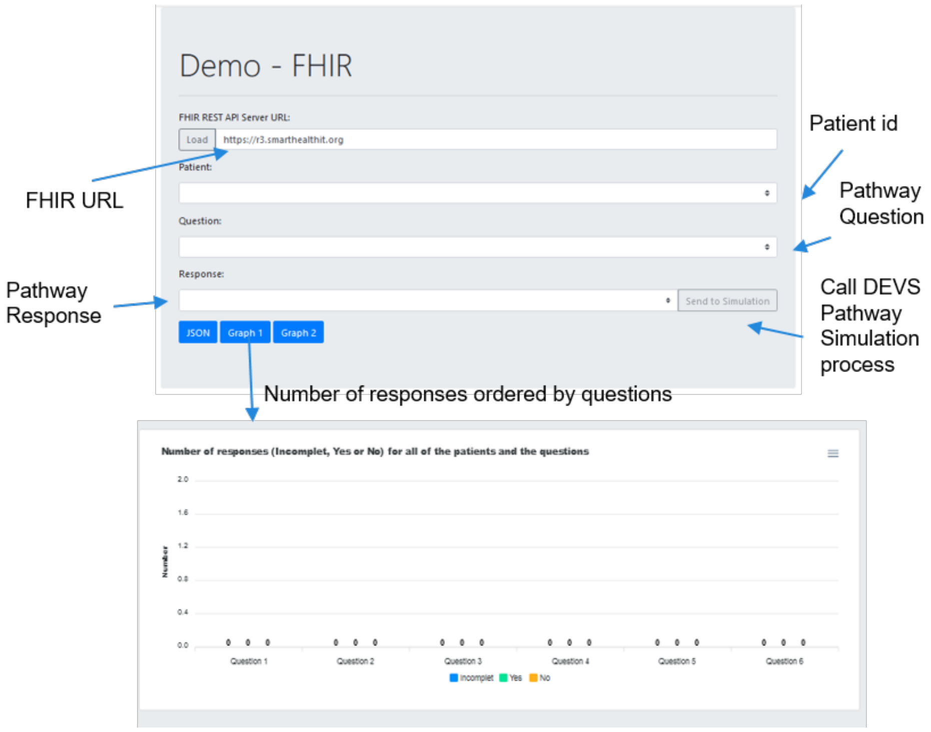
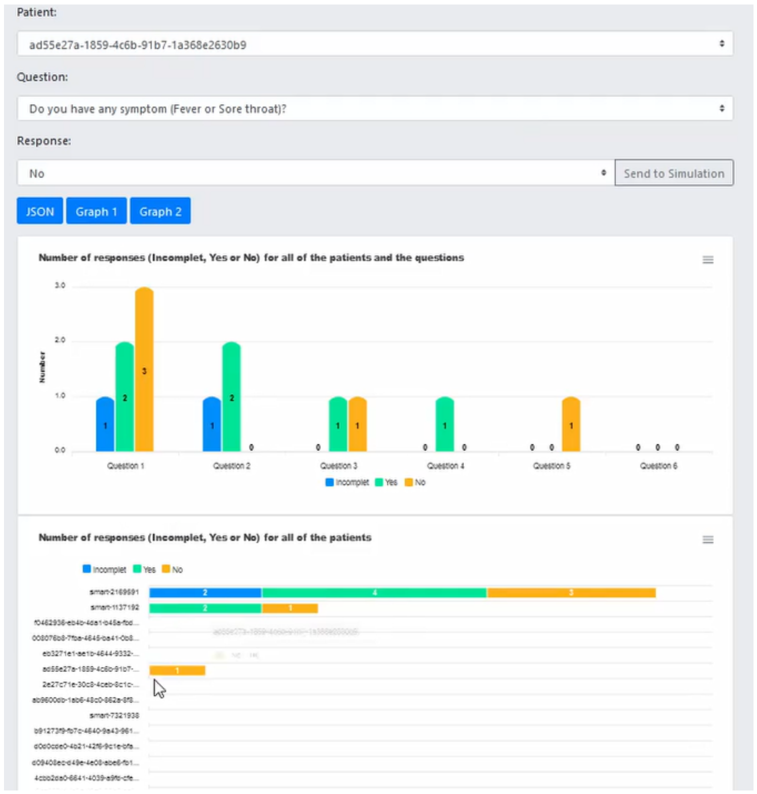

Demo - FHIR
FHIR REST API Server URL:
Load
https://r3.smarthealthit.org
Patient:
Question:
Response:
Send to Simulation
JSON
Graph 1
Graph 2
FHIR URL
Patient id
Pathway Question
Pathway Response
Call DEVS Pathway Simulation process
Number of responses ordered by questions
Number of responses (Incomplet, Yes or No) for all of the patients and the questions
Number
Question 1
Question 2
Question 3
Question 4
Question 5
Question 6
Incomplet
Yes
No

Figure 8: The web app for the end user allowing the communication with the DEVS simulation model.

5.2 Web Interface Design and DEVS Simulation

Figure 8 shows the web app built by the engineer for the end user to communicate with the previous DEVS simulation model. The URL of FHIR (Fast Healthcare Interoperability Resources) (Saripalle 2019) is used to load the 50 patient profiles that are used to complete the select input list (Patient id). FHIR is a standard in the world of health that allows one to develop and implement health interoperability more simply and quickly. The FHIR API Rest and Data models represent discrete clinical and administrative units (patient, medication, etc.). The list of questions starts with Question 1 and is extended during the simulation depending on the responses on the path. The ”Send to simulation” button allows sending the tuple question/response (inserted by the end user) to the simulation model that returns the new question if needed. The simulation results are retrieved by receiving a socket message, which is parsed, and the next new question is redirected to the Web App interface. Graphs are introduced to track application statistics


Patient: ad55e27a-1859-4c6b-91b7-1a368e2630b9

Question: Do you have any symptom (Fever or Sore throat)?

Response: No

Send to Simulation
JSON
Graph 1
Graph 2
Number of responses (Incomplet, Yes or No) for all of the patients and the questions
Number
3.0
2.0
1.0
0.0
Question 1
Question 2
Question 3
Question 4
Question 5
Question 6
Incomplet
Yes
No
Number of responses (Incomplet, Yes or No) for all of the patients
smart-2169691
smart-1137192
f0462936-eb4b-4da1-b45a-fbd...
008076b8-7fba-4645-ba41-0b8...
eb3271e1-ae1b-4644-9332-...
ad55e27a-1859-4c6b-91b7-...
2e27c71e-30c8-4ceb-8c1c-...
ab9600db-1ab6-48c0-862a-8f8...
smart-7321938
b91273f9-fb7c-4640-9a43-961...
d0d0cde0-4b21-42f6-9c1e-bfa...
d09408ec-d49e-4e08-abe6-fb1...
4cbb2da0-6641-4039-a9fb-cfe...

Figure 9: Case study of an interaction for the patient *ad55e27a-1859-4c5b-91b7-1a368e2630b9* with the question ”Do you have any symptom (Fever or Sore Throat)?” and the response ”No”.

and are added by inserting HTML code outside of the Web App Builder algorithm. In figure 9, graph 1 shows the number of response values (incomplete, yes or no) by questions, while graph 2 shows the number of response values by patients. The select input list, the button ”Send to Simulation” and the list of questions are generated automatically from the yaml file as explained previously (see Web App Builder algorithm). The engineer implements all other input and output fields manually using the drag-and-drop functionality from the Web App Builder Interface and by coding HTML tags. HTML and JavaScript files have been edited to implement the event handler for the JSON, Graph1, and Graph2 buttons.

6 CONCLUSION

This article presents an approach for performing the DEVS model simulation using a web application. The approach allows a user to generate a web application associated with a given DEVS model to simulate. A set of DEVS-based web building blocks are provided in order to allow an user to start a simulation of a given DEVS model from a web app, to populate the model through the web app, and to display the results on the web app. Furthermore, a set of basic design elements of a web application have been defined in order to guide the user to automatically generate a web application associated with a given DEVS model. The presented approach is based on the use of: (i) DEVSimPy Design Block/Pattern concepts and implementation; (ii) Asynchronous HTTP Client/Server (AIOHttp), and (iii) Object network communication (Pyro4). It has been validated in a case study related to a FHIR server of the COVID-19 epidemic.

Future work will concentrate on a number of investigations, which are briefly described as follows: (i) to improve the proposed system by managing user accounts, storing user data and models, and providing information of the different user simulations. Users will be able to communicate with each other directly and will have the ability to connect to the same DEVS simulation model and (ii) to connect DEVSimPy with other simulation web services (DEVS-suite simulator (Fard and Sarjoughian 2021) for example) or with other model stores (MS4Me (Seo et al. 2013) for example).

ACKNOWLEDGMENTS

This research was partially supported by RTSync Corporation. We thank the company that provides the funding for the undergraduate student involved in this research work.

REFERENCES

Al-Zoubi, K., and G. Wainer. 2015. "Distributed Simulation of DEVS and Cell-DEVS Models Using the RISE Middleware". *Simulation Modelling Practice and Theory* 55:27–45.

Bankar, S. 2018, June. "Cloud Computing Using Amazon Web Services". *International Journal of Trend in Scientific Research and Development* 2(4):2156–2157.

Belkhir, A., M. Abdellatif, R. Tighilt, N. Moha, Y. Guéhéneuc, and E. Beaudry. 2019. "An Observational Study on the State of REST API Uses in Android Mobile Applications". In *Proceedings of IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems*, edited by K. Moran, 66–75. Los Alamitos, CA, USA: Institute of Electrical and Electronics Engineers, Inc.

Byrne, J., C. Heavey, and P. Byrne. 2010. "A Review of Web-based Simulation and Supporting Tools". *Simulation Modelling Practice and Theory* 18(3):253–276.

Capocchi, L., and J.-F. Santucci. 2021. "A Web-based Simulation of Discrete-event System of System With the Mobile Application DEVSimPy-mob". *SoftwareX* 13. No. 100625.

Capocchi, L., J. F. Santucci, B. Poggi, and C. Nicolai. 2011, June. "DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems". In *Proceedings of 20th IEEE International Workshops on Enabling Technologies*, edited by S. Reddy and S. Tata, 170–175. Los Alamitos, CA, USA: Institute of Electrical and Electronics Engineers, Inc.

AIOHttp developers. "AIOHttp". https://github.com/aio-libs/aiohttp. accessed 19 April 2022.

PyRo4 Developers. "Pyro4". https://github.com/irmen/Pyro4. accessed 19 April 2022.

Elbattah, M., O. Molloy, and B. P. Zeigler. 2018. "Designing Care Pathways Using Simulation Modeling and Machine Learning". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 1452–1463. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Fard, M. D., and H. S. Sarjoughian. 2021. "A Restful Persistent Devs-Based Interaction Model For The Componentized Weap and Leap Restful Frameworks". In *Proceedings of the 2021 Winter Simulation Conference*, edited by S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, 1–12. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Gupta, A., P. Goswami, N. Chaudhary, and R. Bansal. 2020. "Deploying an Application using Google Cloud Platform". In *Proceedings of the 2020 2nd International Conference on Innovative Mechanisms for Industry Applications*, 236–239. New York, USA: Institute of Electrical and Electronics Engineers, Inc.

Ha, S., E. Dimitrova, S. Hoops, D. Altarawy, M. Ansariola, D. Deb, J. Glazebrook, R. Hillmer, H. Shahin, F. Katagiri, J. McDowell, M. Megraw, J. Setubal, B. Tyler, and R. Laubenbacher. 2019. "PlantSimLab - A Modeling and Simulation Web Tool for Plant Biologists". *BMC Bioinformatics* 20(1):1–11.

Capocchi L. "DEVSimPy". https://github.com/capocchi/DEVSimPy. accessed 10 October 2019.

Middleton, N., and R. Schneeman. 2013. *Heroku: Up and Running*. 1st ed. Sebastopol, California: O'Reilly Media, Inc.

Nagpal, A., and G. Gabrani. 2019. "Python for Data Analytics, Scientific and Technical Applications". In *Proceedings of the Amity International Conference on Artificial Intelligence*, edited by S. K. Khatri, A. Rana, and P. Kapur, 140–145. New York, USA: Institute of Electrical and Electronics Engineers, Inc.

Risco-Martín, J. L., S. Mittal, J. C. Fabero Jiménez, M. Zapater, and R. Hermida Correa. 2017. "Reconsidering the Performance of DEVS Modeling and Simulation Environments Using the DEVStone Benchmark". *Simulation* 93(6):459–476.

Saripalle, R. K. 2019. "Fast Health Interoperability Resources FHIR: Current Status in the Healthcare System". *International Journal of E-Health and Medical Communications* 10(1):76–93.

Sarjoughian, H. S., and B. Zeigler. 1998. "DEVSJAVA: Basis for a DEVS-based Collaborative M&S Environment". *Simulation Series* 30:29–36.

Segura, S., J. A. Parejo, J. Troya, and A. Ruiz-Cortés. 2018. "Metamorphic Testing of RESTful Web APIs". In *Proceedings of IEEE/ACM 40th International Conference on Software Engineering*, edited by J. Noble and J. Potter, 882–882. New York, NY, USA: Association for Computing Machinery.

Seo, C., and B. P. Zeigler. 2009. "Automating the DEVS Modeling and Simulation Interface to Web Services". In *Proceedings of the 2009 Spring Simulation Multiconference*, edited by G. A. Wainer, C. A. Shaffer, R. M. McGraw, and M. J. Chinni, 1–8. San Diego, CA, USA: Society for Computer Simulation International.

Seo, C., B. P. Zeigler, R. Coop, and D. Kim. 2013. "DEVS Modeling and Simulation Methodology with MS4 Me Software Tool". In *Proceedings of the Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*, edited by G. A. Wainer, P. J. Mosterman, F. J. Barros, and G. Zacharewicz, 33. San Diego, CA, USA: ACM.

Seo, C., B. P. Zeigler, D. Kim, and K. Duncan. 2015. "Integrating Web-Based Simulation on IT Systems with Finite Probabilistic DEVS". In *Proceedings of the Symposium on Theory of Modeling and Simulation: DEVS Integrative M&S Symposium*, edited by F. Barros, M. H. Wang, H. Prähofer, and X. Hu, 173–180. San Diego, CA, USA: Society for Computer Simulation International.

Shaikh, B., G. Marupilla, M. Wilson, M. L. Blinov, I. I. Moraru, and J. R. Karr. 2021. "RunBioSimulations: an extensible web application that simulates a wide range of computational modeling frameworks, algorithms, and formats". *Nucleic Acids Research* 49(1):597–602.

Singh, H., T. Bhatia, P. Litoria, and B. Pateriya. 2019. "Development of a Web GIS Application Using Web AppBuilder for ArcGIS (Developer Edition)". 10:33–38.

St-Aubin, B., E. Yammine, M. Nayef, and G. Wainer. 2019. "Analytics and Visualization of Spatial Models as a Service". In *Proceedings of the 2019 Summer Simulation Conference*, edited by U. Durak, 1–12. San Diego, CA, USA: Society for Computer Simulation International.

Uran, S., and K. Jezernik. 2006. "MATLAB Web Server and M-file Application". In *Proceedings of the 12th International Power Electronics and Motion Control Conference*, 2088–2092.

Wang, S., and G. Wainer. 2016. "Modeling and Simulation as a Service Architecture for Deploying Resources in the Cloud". *International Journal of Modeling, Simulation, and Scientific Computing* 7(1641002):1–38.

Zeigler, B. 2021. "DEVS-Based Building Blocks and Architectural Patterns for Intelligent Hybrid Cyberphysical System Design". *Information* 12(531):1–21.

Zeigler, B., A. Muzy, and E. Kofman. 2018. *Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations*. London, United Kingdom: Elsevier Science.

Zeigler, B. P. 2018. "Discrete Event System Specification Framework for Self-Improving Healthcare Service Systems". *IEEE Systems Journal* 12(1):196–207.

AUTHOR BIOGRAPHIES

LAURENT CAPOCCHI was born in Bastia, Corsica, France. He received a MSc in electrical engineering from "Ecole Superieure d'Ingénieurs de Nice Sophia Antipolis", ESINSA, Nice, France in 2001 and the PhD in computer science from University of Corsica "Pasquale Paoli", Corte, France in 2005. His main research concerns the M&S of complex systems by using discrete-event approach. He is a founding member of the DEVSimPy/DEVSimPy-mob suite open source project and actively contributes to its development. His email address is capocchi@univ-corse.fr and his homepage is https://capocchi-l.universita.corsica/.

JEAN-FRANÇOIS SANTUCCI is Full Professor of Computer Sciences at the University of Corsica (France), SPE CNRS 6134 Research Lab since 1995. He was Assistant Professor at the EERIE School, Nimes, France, from 1987 to 1995. He has been the author or coauthor of more than 150 papers published in international journals or conference proceedings. Furthermore, he has been the advisor or co-advisor of more than 30 PhD students and since 1998 he has been involved in the organization of ten international conferences. His main research topics include M&S, development of new concepts around the DEVS Formalism, Test pattern generation from behavioral descriptions, and improvement of Reinforcement Learning using M&S. His email address is santucci@univ-corse.fr.

BERNARD P. ZEIGLER is Professor Emeritus of Electrical and Computer Engineering at the University of Arizona (USA) and Chief Scientist of RTSync Corp. (USA). Dr. Zeigler is a Fellow of IEEE and SCS and received the INFORMS Lifetime Achievement Award. He is a co-director of the Arizona Center of Integrative M&S. His research interests include theory of M&S, parallel and distributed simulation, and model construction methodology. His email address is zeigler@rtsync.com.

JOHANNA FERICEAN is a graduate student of Master's degree in Computer Science - Full Stack Development at the University of Corsica "Pasquale Paoli". She studied discrete-event M&S Theory and more specially the DEVS formalism. It helps to maintain the DEVSimPy environment with which she has implemented all of its discrete-event models and plugins. Her email address is johanna.fericean@webmail.univ-corse.fr.