Proceedings of the 2017 Winter Simulation Conference
W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, eds.

GENERIC ARCHITECTURE FOR INTERACTIVE MOBILE SIMULATION OF PARALLEL DEVS MODELS: A MISSILE DEFENSE APPLICATION

Celine Kessler
SPE UMR CNRS 6134
University of Corsica
Campus Grimaldi,
Corte, 20250 FRANCE

Laurent Capocchi
SPE UMR CNRS 6134
University of Corsica
Campus Grimaldi,
Corte, 20250 FRANCE

Bernard P. Zeigler
University of Arizona
Arizona Center for Integrative M&S
RTSync Corp.
Potomac, MD, 20854 USA

Jean-François Santucci
SPE UMR CNRS 6134
University of Corsica
Campus Grimaldi
Corte, 20250 FRANCE

ABSTRACT

Modeling and simulation (M&S) is a discipline oriented towards engineering and research, but it tends since the very last years to be used more and more by users and developers of mobile applications through cloud computing and web services. The M&S new tools involve mobile terminals (smartphone, tablet, etc.) exchanging data quantities increasingly important from sensors with an increasing transmission speed. This paper presents a generic approach (the DEVSimPy-mob mobile application) which aims to simulate models described with the DEVS formalism (Discrete EVent system Specification). DEVSimPy-mob communicates with a web REST (Representational State Transfer) server that delivers a set of web services dedicated to the simulation of DEVS models. A real case application stemming from Balistic Missile Defense simulations is presented to show how DEVSimPy-mob can be used to launch simulations from a mobile device, interact during the simulation process and visualize results.

1. INTRODUCTION

Modeling and simulation (M&S) is a discipline first of all oriented towards engineering and research, but it tends since the very last years to be used more and more by users and developers of mobile applications through cloud computing and web services. The advantage of using simulation tools via web services is not new. In (Page et al. 2000) the authors question the potential impact of using these services with respect to the modeling methodology that is used. They conclude by noting that the combination of web and simulation surely lead to change our approach to the modeling of complex systems in the future. On the other hand, in (Taylor et al. 2013) the authors emphasize the importance of fashion and simulation based on the use of web services but also the arrival of ubiquitous systems such as Smartphones, tablets, etc.. Of course this introduces the issues involved in the real time interaction of simulation tools at the user level. Nowadays, it is obvious that tools and approaches are proposed in order to model and simulate ubiquitous systems through the intermediary of web services. These tools and approaches make it possible to integrate the simulation as a service accessible by mobile devices (smartphones) or to integrate mobile devices (or components embedding sensors) as a source of data for the simulation (Campillo-Sanchez et al. 2013).
A set of questions involved by the connection between simulation, cloud and smartphones has been raised in (Taylor et al. 2013):

- How to interface simulation software with Smartphone APIs?
- How to combine discrete-event simulation, cloud computing (with web services interfaces) and mobile devices?

- How to use a more abstract approach to deal with these problems?

We need to explore new architectures, including hybrid approaches suitable for handheld devices. We also need to explore new mechanisms to allow users to collaborate on joint simulation exercises, sharing information and data between existing devices and the simulation server. We can point out that M&S techniques like DEVS (Discrete Event System Specification) (Zeigler et al. 2000), promise better success by addressing these issues at a higher level of abstraction. DEVS (Zeigler et al. 2000) has been introduced as an abstract formalism for the modeling of discrete event systems, and allows a complete independence from the simulator using the notion of abstract simulator. With DEVS, models, simulators and experiments are systematically built and interoperability can be enhanced.

In this paper we describe a generic approach allowing to: (i) simulate DEVS models using smartphones; (ii) interact during the simulation from the smartphone; (iii) visualize results on the smartphone.

DEVSimPy (Capocchi et al. 2011) is a general user interface dedicated to modeling and simulate DEVS models. It is based on the API Py(P)DEVS for the DEVS simulation kernel which is supported by the Modeling, Simulation and Design lab (MSDL) of the McGill University. We propose a new mobile app (DEVSImPy-mob) to simulate DEVSimPy models in a remote way via web services. The aim of DEVSimPy-mob is primarily to give users the option of executing DEVSimPy models from mobile devices. The mobile becomes a input data source for simulated models and allows the user to contextualize its simulations. Indeed, initially the user can select a model based on its position or the context in which it is located (mobility). So the selected model is dependent on contextual data that may be used by the simulation. Finally, the simulation can be executed from contextual data that are likely to influence the choice of simulation algorithms (sequential, parallel or distributed).

The rest of the paper is organized as follows. The next section gives the context of the work: a state-of-the-art of simulation via smartphone is proposed before presenting the main concepts involved in the DEVS formalism and the DEVSimPy/DEVSimPy-mob environment. The generic approach allowing to define mobile applications to perform simulations while the validation of the approach is conducted using a Balistic Missile Defense System (BMDS) application is described in section 4. The last section is devoted to concluding remarks and future work.

2. CONTEXT: DEVS AND DEVSIMPY

The classic DEVS (Discrete EVent system Specification) (Zeigler et al. 2000) formalism has been introduced as an mathematical abstract formalism for the modeling and the simulation of discrete-event systems allowing a complete independence from the simulator using the notion of abstract simulator. DEVS defines two kinds of models: atomic and coupled models. An atomic model is a basic model with specifications for the dynamics of the model. It describes the behavior of a component, which is indivisible, in a timed state transition level. Coupled models tell how to couple several component models together to form a new model. This kind of model can be employed as a component in a larger coupled model, thus giving rise to the construction of complex models in a hierarchical fashion. As in general systems theory, a DEVS model contains a set of states and transition functions that are triggered by the simulator.

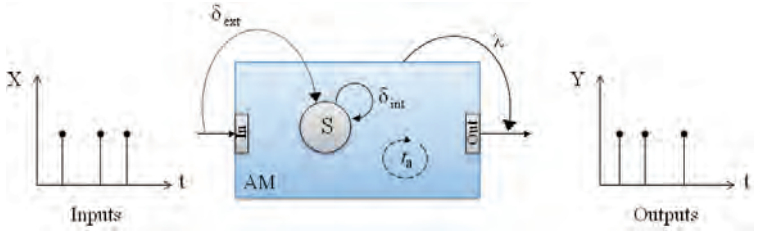


Figure 1: DEVS classic atomic model in action.

The Figure 1 describes the behavior of a discrete-event system as a sequence of deterministic transitions between sequential states (S). The atomic model AM reacts depending on two types of events:

external and internal events. When an input event occurs (X), an external event (coming from another model) triggers the external transition function δext(X,S) of the atomic model in order to update its state. If no input event occurs, an internal event triggers the internal transition δint(S) of the atomic modeling order to update its state. Then, the output function λ(S) is executed to generate the outputs (Y). ta(S) is the time advance function which determine the life time of a state.
Parallel DEVS (DEVS) (Chow and Zeigler 1994) has been introduced as an extension of Classic DEVS, to provide a formalism that take better account the parallelism by introducing the notion of bags, the confluent transition function. Indeed, with bags, the external transition function receives multiple messages on a single port. The new confluent function is triggered when a model receives an external input at the same time as it would do its own internal transition. It can be notice that, the output function generates now a bag instead of a single output.
In some situation, DEVS coupled models need to change their coupling or/and their composition during the simulation. Dynamic Structure DEVS (DSDEVS) formalism (Barros, 1995) introduces a network executive in each coupled model that can receive messages from all of its components to initiate a restructuring. A practical approach to dynamic structure has been proposed in (Hu et al.,2000) while recent application of dynamic structure DEVS can be mentioned (Steiniger and Urhmacher 2016, Muzy and Zeigler 2017).

DEVSimPy (Capocchi et al. 2011) is an Open Source project (under GPL V.3 license) supported by the SPE (Science pour l'Environnement) group of the UMR CNRS 6134 Lab. of the University of Corsica "Pasquale Paoli". This aim is to provide a GUI for the M&S of PyDEVS and PyPDEVS (Li et al. 2011) models. PyDEVS is an Application Programming Interface (API) allowing the implementation of the DEVS formalism in Python language (Perez et al. 2011). PyPDEVS is the parallel version of PyDEVS based on Parallel DEVS formalism (Chow and Zeigler 1994). The DEVSimPy environment has been developed in Python with the wxPython (Rappin and Dunn 2006) graphical library without strong dependencies other than the Scipy (Jones et al. 2001) and the Numpy (Oliphant 2007) scientific python libraries. The basic idea behind DEVSimPy is to wrap the PyDEVS API with a GUI allowing significant simplification of handling PyDEVS/PyPDEVS (Van Tendeloo 2014) models (like the coupling between models or their storage into libraries).

3. GENERIC PROPOSED APPROACH

The section points out how we dealt with the three main questions introduced in the introduction: (i) how to interface simulation software with Smartphone APIs; (ii) How to combine discrete-event simulation, cloud computing (with web services interfaces) and mobile devices; (iii) How to use a more abstract approach to deal with these problems.

The proposed approach is based on the use of a new mobile app DEVSimPy-mob dedicated to remote simulation of DEVS models which have been implemented from the DEVSimPy environment.

In order to propose an abstract/generic as pointed in question (iii), DEVSimPy-mob is based on a client/server architecture where the client (mobile device) is connected to the DEVSimPy-REST (Representational State Transfer) server which operates command line version of DEVSimPy : DEVSimPy-nogui (Figure 2).

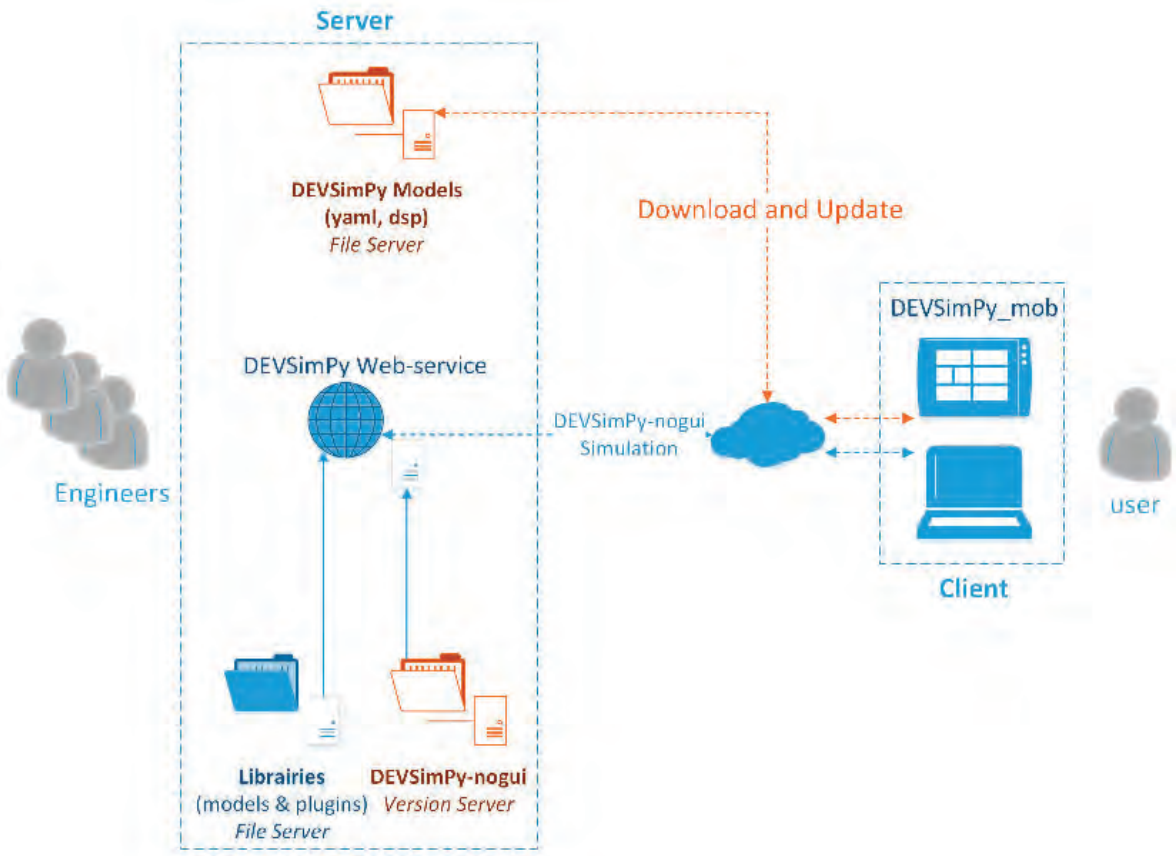


Figure 2: Client/Server architecture of DEVSimPy-nogui.

Developers build and validate their DEVS models in the .yaml (YAML Is not Markup Language) format using the DEVSimPy environment. These models are synchronized to a remote file server and they are accessible by the DEVSimPy-REST server. In the same action, they develop models libraries (with DEVSimPy specific .amd or .cmd extensions) needed for .yaml models. These libraries are synchronized to another remote file server to be shared between developers. Users can connect to the server via an URL and the list of available .yaml models appears in order to be simulated. The use of the YAML format allows us to interact with the models (read/write to modify the model parameters for example) without being dependent on the DEVSimPy environment in which they were implemented.

To combine discrete event simulations with web services and mobile devices as pointed out in question (ii), the simulations are carried out on the server REST DEVSimPy-nogui in order to benefit from the possible parallelization (or distribution) (depending on the OS of the web server) of the simulation thanks to the PyPDEVS kernel available in DEVSimPy-nogui. The algorithms of the DEVS simulation kernel are not embedded on the mobile terminal. The DEVSimPy-mob application therefore does not offer any disconnected mode and it will not be able to perform simulations if the connection is not established with a DEVSimPy-REST server.

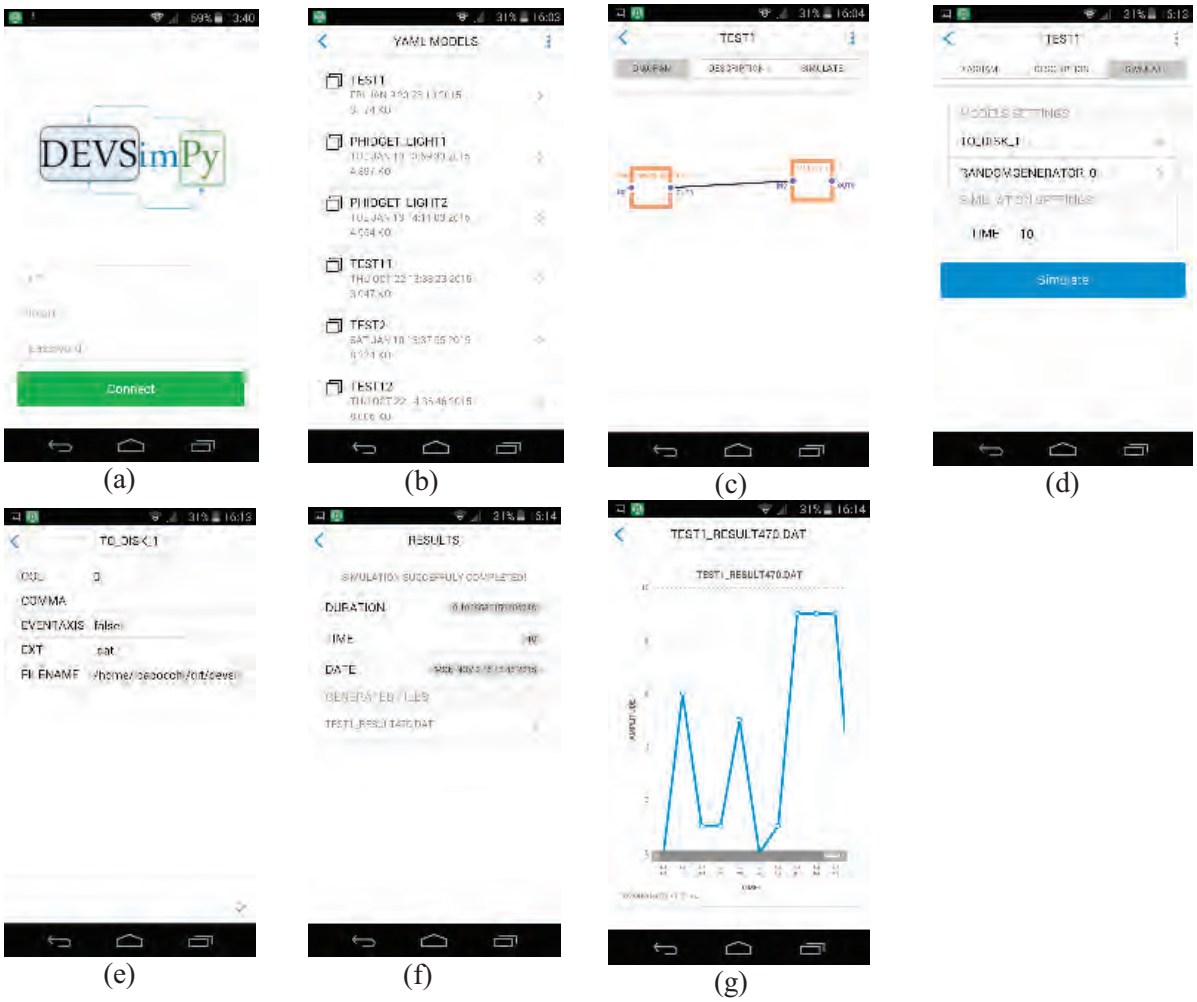


Figure 3: Screenshots of the DEVSimPy-mob mobile interface.

The solution to question (i) is brought by defining the interface of DEVSimPy-mob based on the use of an interactive menu which allows:

1. Connecting to a REST server DEVSimPy-nogui via its URL (Figure 3a).
2. The choice of a model stored on the REST server after the connection (Figure 3b).
3. Visualization of the model chosen for the simulation (Figure 3c).
4. Simulation settings (Figure 3d).
5. The model settings (Figure 3e). The validation of the changes will update the YAML files.
6. Confirmation of the end of the simulation with access to the results (Figure 3f). The time spent for the simulation on the server is also given in seconds.
7. Displaying the graphic of the simulation results (Figure 3g).

4. CASE STUDY: BALISTIC MISSILE DEFENSE APPLICATION

4.1 Description

We have validated the proposed approach of section 3 using a real case problem stemming from problematics raised in the framework of Ballistic Missile Defense System (BMDS) in the United States.

The example concerns the study of a system involving a set of components (mainly missiles, sensors and interceptors). The study requires M&S to synchronize sustained acquisitions of this component system. The M&S is used to compute how these three kinds of components are interacting:

- Missiles which update every time step until entering target zone or being intercepted.
- Sensors which receive updates from all missiles but track only their assigned missile - they receive a notice their assigned missile lands and it can tell if the missile has been intercepted.
- Interceptors which have a flight period after which they disabled any missile within its kill zone - a circle with selected radius-.

The number of missiles equals the number of sensors. The number of interceptors is unconstrained. Missiles and Interceptors are independently distributed in a 2-D space.

Figure 4 describes the simulation process involving missiles and sensors.
Interceptors can interact with missiles by avoiding the landing of missiles if they succeed in intercepting missiles by entering the kill zone of a missile (missile intercepted) before the missile reach the target zone (missile landed).

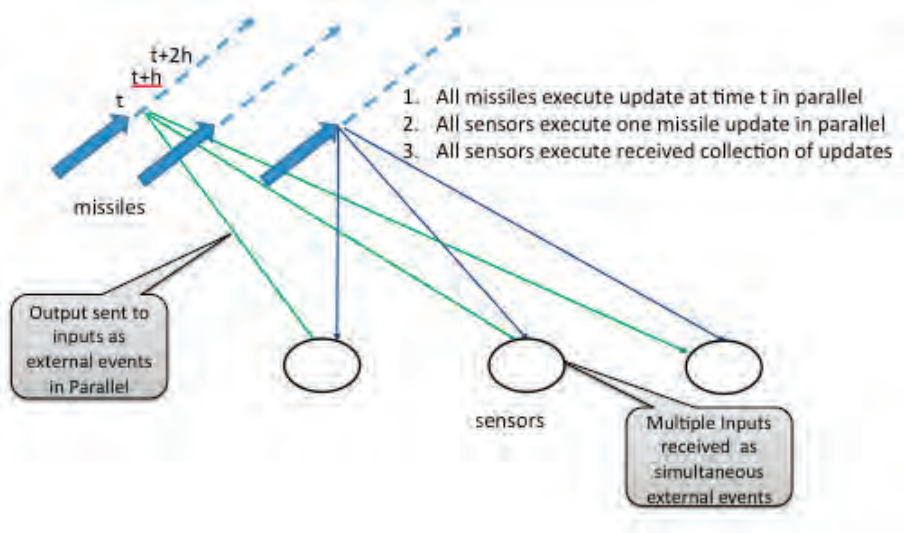


Figure 4: Simulation of missiles and sensors.

The goal is to be able: (i) to start a simulation of the system involving missiles, sensors and interceptors using mobile application; (ii) to eventually add interceptors during the simulation; (3) to visualize on the phone the information concerning the landing of missiles, the number of missiles intercepted, etc…
The choice of using the BMDS as an application has been motivated by the fact that during the simulation the structure of the simulated models are dynamically changed by a simple push on a button of a smartphone. Of course the results of simulations including these structure modifications are visualized on the smartphone screen. The BMDS is a good example for illustrating the combination of simulations with mobile application as it illustrates the need to interact with the simulation by immediately modifying structure (by adding interceptors during the simulation process) and to see results on the mobile device.

4.2 DEVSimPy Implementation

In this section we describe the DEVS and DEVSimPy implementation of the missile application. We first present the different atomic models involved in the modeling phase of the component system. We then detail how using variable dynamic structure involved in the DEVS formalism extensions we are able to add interceptor components during the simulation process. Finally we give the coupled model allowing to link sensors, missiles and interceptors and the corresponding results.

4.2.1 Atomic Models

The Missile atomic model has one input and one output. The state set is composed by the two usual DEVS state variables: the variable "Phase" with four potential values ("Update", "Intercept", "InTargetZone", "Repeat") and the variable "Sigma" representing the lifetime in a given "Phase".
The model computes an output value corresponding to the actual state of the modeled missile (Landed, Updated or Intercepted). The computation is performed according to the input message which comes from the interceptor atomic models, which give the location of the corresponding interceptors.

A Sensor atomic model has one input and one output. The state set is composed by the two usual state variables : "Phase" with three potential values ("GetUpdate", "LostTrack", "SendInfo") and the variable "Sigma" representing the lifetime in a given "Phase". The model computes an output value corresponding to the actual state of the modeled sensor ("Intercept", "Tracking", "HitTragetZone"). The computation is performed according to the input message which comes from the missile atomic models ("Intercepted", "Updated" or "Landed"). The output message gives for the missile associated with the given sensor, its new state ("Intercept", "Tracking" or "HitTargetZone") according to the input value. We have to point out that the update time period is 2 time unit.
An Interceptor atomic model has one output (and no input so that it can act as a generator). The model computes randomly the location of a potential interception of a missile and this location is sent to the associated missiles atomic models. We may point out that the update time period of the location is 1 time unit.

4.2.2 Coupled Models

We propose two kinds of coupled model: in the first one the number of sensors, missiles and interceptors are predefined and fixed while in the second one the three kinds of atomic models will be added during simulation.

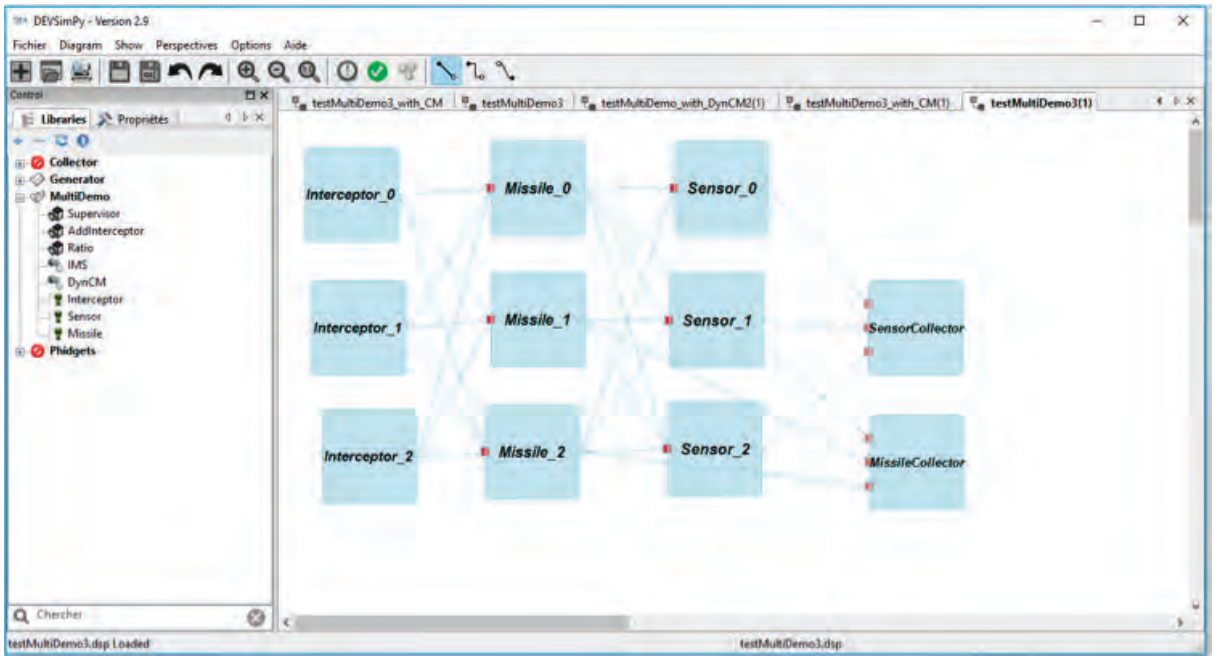


Figure 5: Coupled Model without dynamic variable structure.

The figure 5 gives a coupled model involving three instances of each atomic model (sensors, interceptor

and missile). Two collector atomic models are added to visualize the output of the missile and sensor atomic models.

We propose to improve the definition of a coupled model involving the three kinds of atomic models presented before. The improvement is be proposed in order to: (1) facilitate the definition of the interconnection between the missiles, sensors and interceptors - when for example there are many kinds of components - and (2) offer the possibility to add interceptors dynamically during the simulation process in order to raise the ratio of intercepted missiles.
The user has just to define the number of each kind of atomic models and the models are dynamically added to the initial coupled model (which is initially composed by *To_Disk* models which are used only for results visualization).
Figure 6 shows the initial configuration of coupled model.

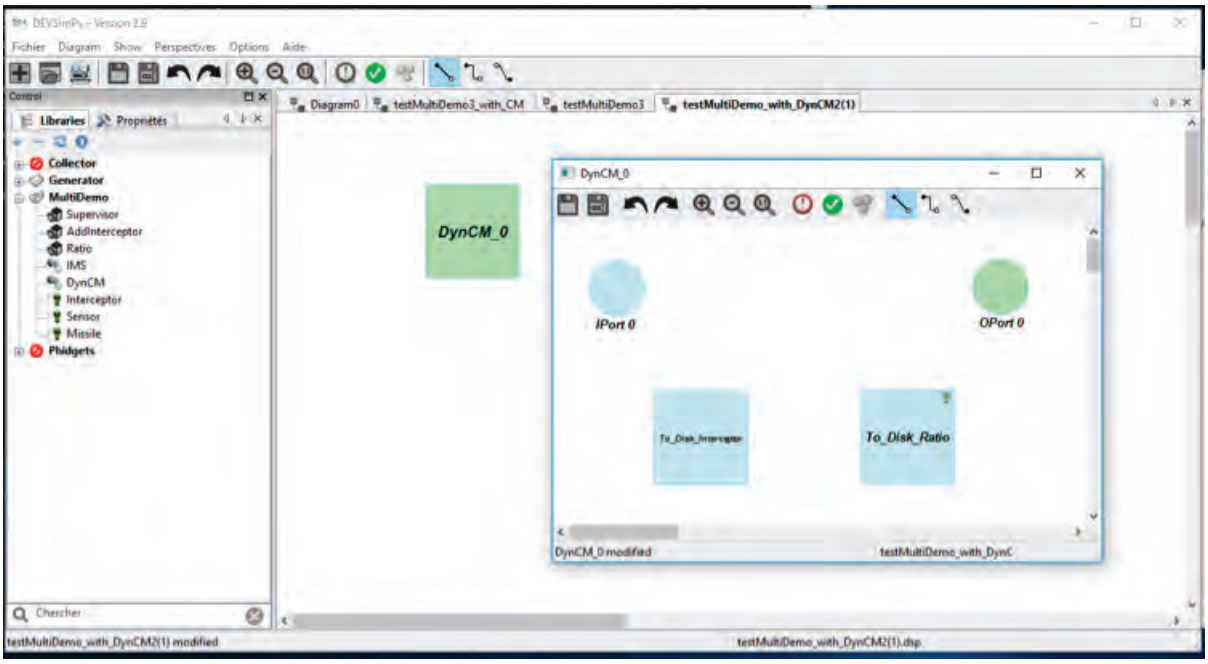


Figure 6: Coupled model with dynamic variable structure.

The way the user can specify the number of missiles, sensors and interceptors is described in figure 7.
The dynamic addition of atomic models and connections is realized using the *DynCM* Class. This coupled model can now cause changes in the structure of the initial model. The constructor of the *DynCM* Class is adding sub-models, adding ports, and connecting ports in order to realize the required interconnections. Such changes happen at run-time.
The way we perform dynamic structure changes is based on the following: after each transition function, the transitioned model will have its *modelTransition(state)* method called (Van Tendeloo 2014). If it returns *True*, the model will signal its parent model that it requests a structural change. The model will then have its *modelTransition(state)* called too, which is allowed to make structural changes. If it requires changes at a higher level, it should return *True* again, to have the structural change request being propagated upwards. If it returns *False*, no change is requested. All structural changes happen in the *DynCM* coupled model.

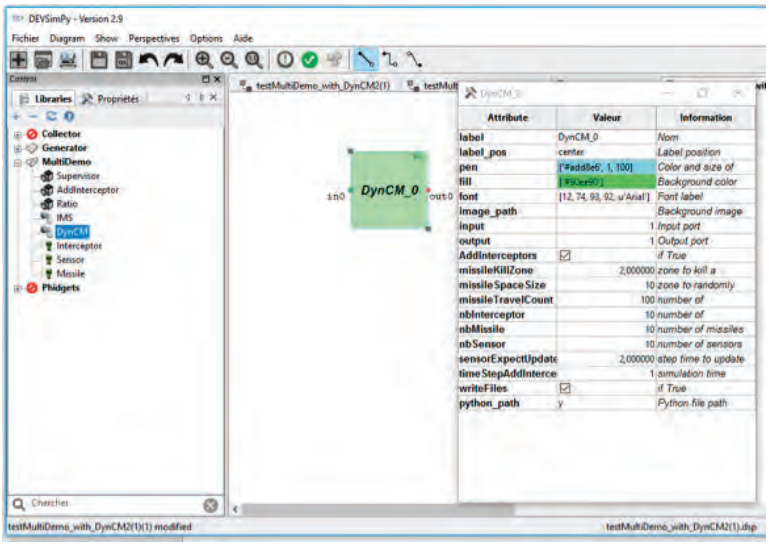


Figure 7: How to specify initial numbers of sensors, missiles and interceptors.

Furthermore, in order to define the possibility to add interceptor atomic models during the simulation process, we implement two new atomic models: *AddInterceptors* and *Ratio* atomic models.
The *Ratio* atomic model has one input and one output: the input is connected to all the sensors models while the output is connected to the *Addinterceptor* atomic model. These connections are realized at run-time using the *modelTransition* method of the Ratio Atomic model which return *True* to the *DynCM* coupled parent model in order to perform the structural changes. The changes are required only when the ratio is smaller than 1 (that is to say when some of the missiles not already landed are still not intercepted). The *Addinterceptor* atomic model has one input and one output: the input is connected to the *Ratio* atomic model while the output is connected to the *To_Disk* models which allows to visualize the results (the number of interceptors involved during the simulation process).

4.2.3 DEVSimPy Simulation Results

In this section we detailed the results obtained using the *DynCM* coupled model. Figure 8 presents the interface allowing the user to launch the Dynamic Structure DEVS simulator (he has to choose PyPDEVS instead of the classical PyDEVS simulation and choose the Dynamic Structure option which will allow to enable the *modelTransition* method utilization).

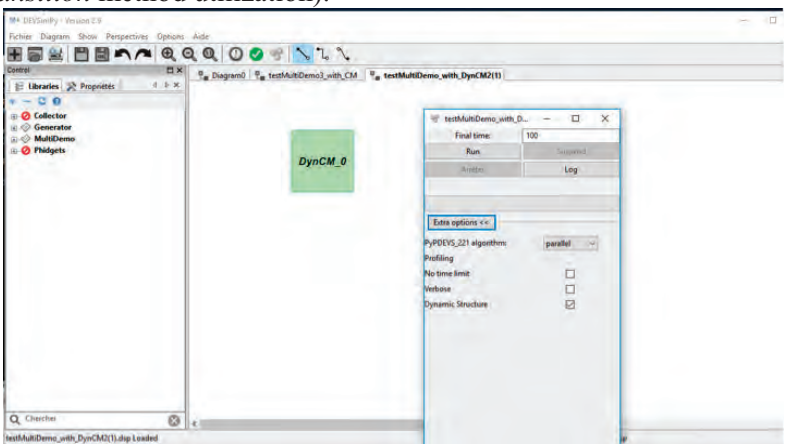
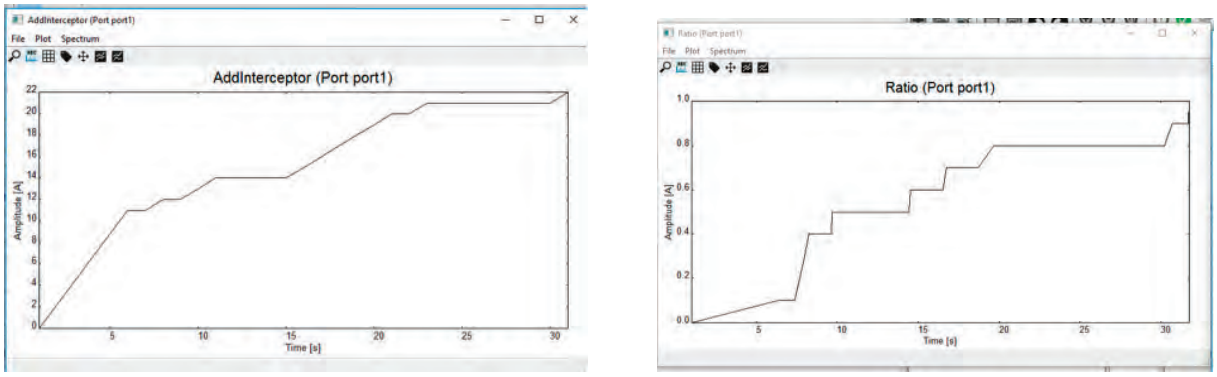


Figure 8: Interface allowing to launch the dynamic variable simulation of *DynCM.*

The results can be visualized using the two *To_Disk* atomic models.
Figure 9a shows how the number of interceptors changes during the simulation while figure 9b shows how the ratio of intercepted missiles on total number of missiles changes during the simulation.

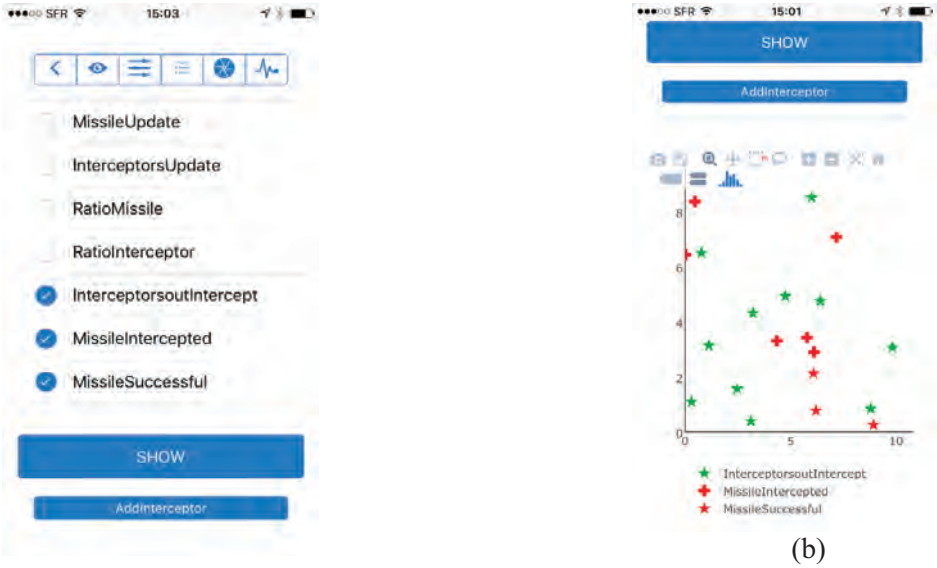


(a) (b)

Figure 9: Simulation results – (a) Numbers interceptors ; (b) Ratio missiles intercepted.

4.3 Missile Interceptor Mobile Application

In this sub-section we present the results obtained using the approach detailed in section 3 in order to run the previously presented model through the ionic view mobile application.
The mobile application DEVSimPy-mob allows to: (1) select and load a selected model from a list of models proposed by the server; (2) initiate remote simulations on the server; (3) visualize and exploit the simulation results in order to stop a decision-making process, for example.
First of all the user has to select the coupled model he wants to simulate. In our case it is the DYNAMINTERACTPUSHER. The user can then perform simulations and finally visualize and exploit the simulation results. For that the user can select the data which have to be visualized on the screen. He can also push on the Addinterceptor button which will allow to add dynamically interceptors to the model. Figure 10a points out this new window obtained after selecting the coupled model as shown previously.



(a) (b)

Figure 10: (a) Visualization: Selection of items – (b) Items Display.

The visualization is performed by the user by clicking on the button SHOW which will allow to obtain the results presented in figure 10b.

5. CONCLUSION AND FUTURE WORK

In this paper we present a progressive mobile application allowing the simulation of DEVS models from a mobile phone. The idea is to allow users of DEVS software frameworks as DEVSimPy to simulate their models from a mobile phone. The user can also interact dynamically with the application from the mobile phone during the simulation process. The mobile application is a part of a client/server solution and must be coupled with a DEVSimPy-REST server which is in charge to simulate *DEVSimPy* models. We have chosen the Ionic web framework in order to make a robust development of DEVS mobile applications. A real case application concerning the Ballistic Missile Defense System in the United States is detailed in order to illustrate the proposed approach. The future work concerns the combination of the *DEVSimPy* environment and the DEVS mobile applications in order to deal with the modeling of ubiquitous systems and their simulation from a mobile phone. Users will be able to manage real data stemming from the mobile phone (temperature, photo, etc) and data coming from sensors in their simulation models. We plan also to use the proposed web-based service simulation for the validation of ubiquitous systems by associating sensors, actuators and embedded boards to DEVS models in a dynamically evolving environment. DEVSimPy-mob will be considered in this case as a collector of real data that are injected in DEVS simulations to perform the validation of the object connected communication. Finally DEVSimPy-Mob will be able to simulation models that are based on machine Learning (Markov models, Neural networks (Toma et al. 2011), Fuzzy logic (Santucci and Capocchi 2014), etc.) in order to perform decision making processes in a remote way.

REFERENCES

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2000. *Discrete-Event System Simulation*. 3rd ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Barros., J.F, 1995, "Dynamic Structure Discrete Event System Specification: A New Formalism for Dynamic Structure Modeling and Simulation". *In Proc. of the 1995 Winter Simulation Conference*, edited by C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldman, 781–785, Arlington, Virginia, USA, IEEE Computer Society Washington.

Chow, A.C.H and Zeigler, B.P. 1994. "Parallel DEVS: A Parallel, Hierarchical, Modular, Modeling Formalism". *In Proc. of the 1994 Winter Simulation Conference*, edited by J. D. Tew, S. Manivannan, . A. Sadowski, and A. F. Seila, 716–722, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pablo Campillo-Sanchez, Emilio Serrano and Juan A. Botía. 2013. Testing Context-aware Services Based on Smartphones by Agent Based Social Simulation. J. Ambient Intell. Smart Environ., vol. 5, no. 3, 311–330, May.

Capocchi, L., J. F. Santucci, B. Poggi, and C. Nicolai. 2011. "DEVSimPy: A Collaborative Python Software for Modeling and Simulation of DEVS Systems". *In Proc. of WETICE*, 170-175 IEEE Computer Society.

Jones, E., Oliphant, T., and Peterson, P. 2001. Scipy: Open Source Scientific Tools for Python. URL: http://www.scipy.org/Citing SciPy [Retrieved: February, 2014].

Hu, X., Zeigler, B.P., and Mittal, S. 2005. "Variable Structure in DEVS Component-based Modeling and Simulation", Simulation, vol. 81, 91–102.

Li, X., Vangheluwe, H., Lei, Y., Song, H., and Wang, W. 2011. "A Testing Framework for DEVS Formalism Implementations. *In Proc. of the 2011 Symposium on Theory of Modeling &*

Simulation: DEVS Integrative M&S Symposium, 183–188, San Diego, CA, USA. Society for Computer Simulation International.

Lopes, J. a., Souza, R., and Geyer, C. 2014. A Middleware Architecture for Dynamic Adaptation in Ubiquitous Computing. Journal of Universal Computer Science, 20:1357–1351.

Muzy,A. and Zeigler, B.P. 2017. Dynamic Structure Formalism Framework for Parallel Discrete Event Specification of Dynamic Systems. *To be published in ACM Trans. Model. Comput. Simul.*

Oliphant, T. E. 2007. Python for scientific computing. Computing in Science and Engineering, 9:10–20.

Perez, F., Granger, B. E., and Hunter, J. D. 2011. Python: An ecosystem for scientific computing. Computing in Science and Engineering, (2):13–21.

Rappin, N. and Dunn, R. 2006. WxPython in action. Manning.

Page, E. H., A. Buss, P. A. Fishwick, K. J. Healy, R. E. Nance, and R. J. Paul. 2000, January. Web-based Simulation: Revolution or Evolution?. ACM Trans. Model. Comput. Simul. 10 (1): 3–17.

Santucci, J.F., Capocchi, L., 2014. "Fuzzy Discrete-Event Systems Modeling and Simulation with Fuzzy Control Language and DEVS Formalism", *In Proc. of SIMUL2014*, Oct. 12-16, Nice, 250-255.

Steiniger A. and Uhrmacher, A.M.. 2016. "Intensional Couplings in Variable-Structure Models: An Exploration Based on Multilevel-DEVS". *ACM Trans. Model. Comput. Simul.* 26, 2, Art. 9, 27 pages.

Taylor, S.J., Khan, A., Morse, K.L., Tolk, A.,Yilmaz, L. and Zander, J. 2013. "Grand Challenges on the Theory of Modeling and Simulation". *In Proc. of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*, 34 :1–34 :8, San Diego, CA, USA. Society for Computer Simulation International.

Toma, S., Capocchi, L., Federici, D., 2011. "A New DEVS-Based Generic Artificial Neural Network Modeling Approach", *In Proc. of the 23rd European Modeling and Simulation Symposium (Simulation in Industry)*, Rome (Italy), Sept. 12-14, 351-356.

Van Tendeloo, Y., Activity-aware DEVS simulation Thesis. UNIVERSITEIT ANTWERPEN , 2014.

Zeigler, B. P., Kim, T.G. and Praehofer, H. 2000. Theory of Modeling and Simulation. 2nd ed. Orlando, FL, USA: Academic Press, Inc.

AUTHOR BIOGRAPHIES

CELINE KESSLER is PhD Student at the University of Corsica. Her email address is kessler.celine@orange.fr.

LAURENT CAPOCCHI is Associated Professor in Computer Science at the University of Corsica since 2007. His main research concerns the modeling and simulation of complex systems using DEVS formalism. He is founder of the DEVSimPy project. His email is capocchi@univ-corse.fr.

BERNAR P. ZEIGLER is Professor Emeritus of Electrical and Computer Engineering at the University of Arizona. While in Arizona, Dr. Zeigler served as the Co-Director of the Arizona Center for Integrative Modeling and Simulation (ACIMS). He is currently Chief Scientist with RTSync Corp. an ACIMS spin-off dedicated to commercialization of ACIMS technology. Zeigler is best known for his highly cited publication, “Theory of Modeling and Simulation” and has received much recognition for his various scholarly publications, achievements, and professional service. He received Lifetime Achievement Awards from both the SCS and the INFORMS Simulation Society. His email is zeigler@rtsync.com.

JEAN FRANÇOIS SANTUCCI is Full Professor in Computer Science at the University of Corsica since 1996. His main research interests concern Modeling and Simulation of complex systems. He has been author or co-author of more than 180 papers published in international journals or conference proceed- ings. Furthermore he has been the advisor or co-advisor of more than 25 PhD students. His email is santucci@univ-corse.fr.